Scheduling the LZV Cup using Answer Set Programming and SWI-Prolog

COSC2780/2973 Intelligent Decision Making

Semester 1, 2023

Instructor: Prof. Sebastian Sardina

Denster Joseph Frank

s3894695@student.rmit.edu.au

Jyoti.

s3880522@student.rmit.edu.au

Srujan Basavaraj

s3856311@student.rmit.edu.au

June 4, 2023

RMIT University, GPO Box 2476, Melbourne, Victoria 3001, Australia

Abstract

Scheduling matches in sports tournaments is a challenging task that requires balancing multiple constraints and objectives. In this report, we propose two distinct solutions for the scheduling of the LZV Cup tournament using Answer Set Programming (ASP) and SWI-Prolog.

The first solution involves the utilization of ASP to model the tournament scheduling problem and generate optimal schedules that adhere to the specified constraints.The second solution leverages SWI-Prolog, a popular Prolog implementation, to develop a separate scheduling algorithm. we formulate the scheduling problem as a set of rules and constraints, enabling efficient and effective schedule generation.

To evaluate the performance of each solution, we conducted experiments with various parameter values and analyzed the resulting schedules. Through graphical analysis, we assessed factors such as scheduling efficiency, fairness, and overall quality of the tournament schedule for both ASP and SWI-Prolog approaches.

In addition to evaluating the scheduling algorithms, we implemented a comprehensive constraint-checking mechanism using Python. This approach systematically verifies each constraint individually, ensuring that the scheduled games comply with all specified requirements.

The utilization of two different approaches, ASP and SWI-Prolog, along with the constraint-checking mechanism, provides a diverse set of tools for generating optimal tournament schedules that satisfy various constraints. The findings of our research can guide tournament organizers in making informed decisions about match scheduling, leading to fair, efficient, and well-balanced tournaments.

1 Introduction

This report is for the Semester 1 2023 COSC2780/2973 Intelligent Decision-Making project at the Royal Melbourne Institute of Technology (RMIT). The report presents a solution to the scheduling problem of the LZV Cup, a non-professional indoor football league, presents unique challenges that require effective decision-making regarding the timing, location, and sequencing of matches throughout the season. This report explores two separate solutions to the scheduling problem using Answer Set Programming (ASP) and SWI-Prolog.

The primary objective of this report is to provide comprehensive solutions to the LZV Cup scheduling problem by leveraging the capabilities of ASP and SWI-Prolog. ASP offers a declarative approach to problem-solving, allowing us to specify the problem constraints and generate optimal solutions. On the other hand, SWI-Prolog provides a powerful logic programming language for implementing complex algorithms and search strategies.

By exploring two separate solutions using ASP and SWI-Prolog, this report aims to provide a comprehensive analysis of the LZV Cup scheduling problem and demonstrate the effectiveness of these programming paradigms in addressing real-world scheduling challenges in sports.

The research paper titled "Scheduling a non-professional indoor football league: a tabu search-based approach" was referenced as a basis for the implementation. All instances and the generated schedule were obtained from this research publication, ensuring a reliable comparison with the reported results.

The report is organized as follows:

- Section 2 Description of the LZV Cup scheduling problem
  - Detailed explanation of the unique challenges and requirements
  - Review of relevant literature relating to scheduling problems
- Section 3 ASP Solution
  - Overview of methodology for ASP solution
  - Formulating the problem as a logic program and specifying constraints
  - Utilizing the Clingo solver for optimal schedules
  - Explanation of the ASP implementation details and key insights
- Section 4 SWI-Prolog

  - Overview of methodology for SWI-Prolog solution
  - Description of the approach using facts, rules, and predicates
  - Finding feasible solutions and optimization techniques used
  - Explanation of the SWI-Prolog implementation details and key insights
- Section 5 How to Run the code
  - Provides a detailed step-by-step guide on how to run the code
  - Explains the structure of the code and any required dependencies
- Section 6 References

2 Problem description

This project involves the scheduling of LZV (Liefhebbers Zaalvoetbal) Cup matches using intelligent decision making. More specifically, the goal of this work is to develop a system that can schedule LZV (Liefhebbers Zaalvoetbal) Cup matches as double round robin tournaments, under the following constraints [VGS19]:

- Each team plays a home match against each other team exactly once;
- Home team availability's, H_i, are respected;
- Away team availability's, A_i, are respected;
- At least m days between two matches with the same team;
- Each team plays at most one game per day;and
- Each team plays at most 2 games in a period of R_{max} days.

There has been some work on the scheduling of time-constrained and timerelaxed leagues as integer linear programs [Knu10; C¸ A20]. While there exists some work in the scheduling of double round robin tournaments [KL09; NGK14; Kyn+17], most of this work seeks to avoid consecutive home or away games for teams as much as possible.

3 ASP Solution

3.1 Knowledgebase Creation:

Answer Set Programming (ASP) is an one of the approaches for tournament scheduling due to its expressive modeling, constraint satisfaction capabilities, scalability, solution optimality, and flexibility. ASP provides a concise and intuitive representation of scheduling constraints, allowing for accurate specification of requirements. It efficiently handles complex scheduling problems, accommodating large-scale instances with numerous teams, and time slots.

This project focuses on the scheduling of LZV (Liefhebbers Zaalvoetbal) Cup matches as double round-robin tournaments while considering various constraints.Our project methodology for ASP implementation is explained below.

The input instances in the paper contain information about scheduling availability for a league season. The first line in the instances file specifies the total number of slots available for the entire season, while the second line indicates the total number of teams participating in the league. The subsequent lines represent the availability of each team for different slots. In input instances, a "1" signifies that a particular slot is designated as a home game for the corresponding team. On the other hand, a "2" indicates that the slot is part of the forbidden game set for that team, meaning teams are not allowed to play during that slot. Lastly, a "0" denotes that the team can play an away match during the respective slot. For our

research and experimentation, we randomly selected 15 input.txt files out of 53. These selected files have been stored in a folder named "input" for further analysis.

To transform this input data into the required format, a Python code has been implemented (datagen_asp.py). The code reads the data and converts it into the format of "availability (Team, Slot, Value)". Each line in this format represents a team's availability to play in a specific slot, denoted by the value assigned. The value can take on three possibilities: 0, 1, or 2, corresponding to the different availability scenarios. For instance, the entry "availability(1, 200, 2)" would indicate that team 1 is forbidden to play during slot 200.

Generated availability information is stored in a folder named "instances_asp" and serves as a repository for converted availability data, facilitating further analysis or utilization in subsequent processes.

3.2 Predicates for ASP solution (Figure 1):

- The first predicate defines that a "Team" is any entity for which there is an availability predicate with that team in the first argument.
- The second predicate defines that a "Slot" is any entity for which there is an availability predicate with that slot in the second argument.
- The third predicate defines whether a given "Team" plays at "home" in a specific "Slot". It checks the availability predicate with the team in the first argument, slot in the second argument, and 1 in the third argument (indicating home).
- The fourth predicate determines whether a given "Team" plays "away" in a specific "Slot". It checks the availability predicate with the team in the first argument, slot in the second argument, and 0 in the third argument (indicating away).
- The fifth predicate determines whether a given "Team" is "forbidden" to play in a specific "Slot". It checks the availability predicate with the team in the first argument, slot in the second argument, and 2 in the third argument (indicating forbidden).
- The sixth predicate gives the Rmax value for Constraint 5.
- The seventh predicate gives the m value for Constraint 6.

3.3 Approach 1: All Hard Constraints

We implemented the solution in `schedule.lp` where all constraints are made hard. In the given code, the constraints can be classified as hard constraints. Hard constraints are rules or conditions that must be strictly satisfied in order to find a valid solution. In this context, the hard constraints are as follows:

- Constraint 1: Each team plays a home game against each other team at most once. Violation of this constraint would result in a team playing multiple home games against the same opponent.
- Constraint 2: Home team availability is respected. This constraint ensures that a team can only play a home game if they are available.
- Constraint 3: Away team unavailability is respected. This constraint guarantees that a team does not play an away game if they are unavailable.
- Constraint 4: Each team plays at most one game per time slot. This constraint ensures that no team is scheduled to play multiple games simultaneously.
- Constraint 5: Each team plays at most two games in a period of Rmax time slots. This constraint limits the number of games a team can play within a specific period to avoid fatigue or scheduling conflicts.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Predicates for LZV cup tournament
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%This predicate gives whether it is team or not
team(Team):-availability(Team,_,_).
%This predicate gives whether it is slot or not
slot(Slot):-availability(_,Slot,_).
% This predicate gives  whether a team plays in home  or not:
home(Team,Slot):-availability(Team,Slot,1).
% This predicate gives  whether a team plays  away  or not:
away(Team,Slot):-  availability(Team,Slot,0).
% This predicate gives  whether a teams is forbidden to play or not:
forbidden(Team,Slot):-availability(Team,Slot,2).
% R max predicate required for constrain 5
max_R(4).
% M slot predicate required for constrain 6
m_slot(60).
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Figure 1: Predicates for ASP solution

- Constraint 6: There are at least M time slots between two games with the same pair of teams. This constraint introduces a minimum time gap between matches involving the same pair of teams to allow for rest and preparation.

All of these constraints are essential for ensuring fair and efficient scheduling of matches for the LZV Cup tournament.

3.3.1 Detail Understanding of code & hard constraints :

Generating Matches: The given ASP code (Figure 2) represents a rule that generates a set of matches for the LZV cup tournament. Using a set comprehension, it creates matches in the form of match (HomeTeam, AwayTeam, Slot). The rule ensures that only valid slots are considered and enforces that the home team and away team are distinct entities. By considering these constraints and the validity of teams, the code provides an efficient way to generate a well-structured set of matches for the LZV cup tournament.

```
% To generate all matches
{match(H, A, S) :  slot(S)}:- team(H), team(A),H!=A .
```

Figure 2: Generate All Matches

Constraints Specification :

- Constraint 1: This constraint (Figure 3) ensures that there are no match combinations in which the home team H faces the same away team A more than once. If such a combination exists, it will be a rule violation.
  - match$(H, A1, S1)$ represents a match between home team H and away team $A1$, where $S1$ is the specific match slot.
  - match$(H, A2, S2)$ represents another match between home team H and away team $A2$, where $S2$ is another match slot.
  - $S1 \neq S2$ ensures that the matching slots $S1$ and $S2$ are different, referring to distinct matches.

– $A1 = A2$ checks if the away teams $A1$ and $A2$ are the same, indicating that the same away team is involved in both matches.

```
% Constraint 1 -each team plays a home game against each other team at most once
:- match(H, A1, S1), match(H, A2, S2),S1!=S2,A1=A2.
```

Figure 3: Constraint 1

- Constraint 2: This Constraint (Figure 4) respects the availability of the home team as the match is prohibited if the home team is away or forbidden. This constraint help ensure that the scheduling of matches respects the availability constraints of the home teams in the tournament.

  – The first line ensures that if there is a match where a specific home team H is scheduled in a certain slot S, and the availability of the home team for that slot is marked as 0 (representing "away"), then the constraint helps to filter out any answer or solution that violates this condition.

  – The second line ensures that if there is a match where a specific home team H is scheduled in a certain slot S, and the availability of the home team for that slot is marked as 2 (representing "forbidden"), then the constraint helps to filter out any answer or solution that violates this condition.

```
%Constrain 2 -home team availability Hi (i ∈ T ) is respected
:- match(H, _, S), away(H, S).
:- match(H, A, S), forbidden(H,S).
```

Figure 4: Constraint 2

- Constraint 3: The constraint (Figure 5) specifies that the away team is considered unavailable when their availability is marked as forbidden. Conversely, the away team is considered available if their availability allows them to play matches, regardless of whether they are designated as home or away.

  – The given constraint ensures that if there is a match scheduled with a specific away team A in a certain slot S, and the availability of the away team for that slot is marked as 2 (representing "forbidden" or unavailable), then this constraint helps filter out any solution or answer that violates this condition.

```
%constrain 3 -  away team unavailability Ai (i ∈ T ) is respected
:- match(H, A, S), forbidden(A,S).
```

Figure 5: Constraint 3

- Constraint 4: This constraint (Figure 6) ensures that there is only one match per slot, avoiding clashes and violations in the scheduling of matches. These constraints consider different combinations of home teams, away teams, and slots to enforce the desired condition of one match per slot.

  These four logical rules are used to ensure the consistency and uniqueness of matches in each context. Let us break down each rule and explain its purpose:

  – The first rule states that if there exist two matches with the same away team A and the same slot S, but different home teams $H1$ and $H2$, then it is considered a violation. This rule ensures that there is only one match per slot, preventing multiple matches from happening simultaneously.

- The second rule states that if there exist two matches with the same home team H and the same slot S, but different away teams $A1$ and $A2$, then it is considered a violation. This rule also enforces the condition of having only one match per slot, preventing multiple matches involving the same home team from happening simultaneously.
- The third rule states that if there exist two matches where the home team in one match plays like an away team in other matches in the same slot S, then it is considered a violation. This rule ensures that a team cannot have two matches in the same slot, regardless of the opponent.
- The fourth rule states that if there exist two matches where the away team in one match plays like a home team in other matches in the same slot S, then it is considered a violation. This rule ensures that a team cannot have two matches in the same slot, regardless of the opponent.

```
% Constrain 4 - each team plays at most one game per time slot
:- match(H1, A, S1), match(H2, A, S2), H1!=H2, S1=S2. % Only one match per slot
:- match(H, A1, S1), match(H, A2, S2), A1!=A2, S1=S2. % Only one match per slot
:- match(H, _, S1), match(_, H, S2), S1=S2.
:- match(_, A, S1), match(A, _, S2), S1=S2.
```

Figure 6: Constraint 4

- Constraint 5: The code (Figure 7) defines predicates and constraints related to periods, home matches, away matches, and total matches per period. It ensures that the total number of matches for each team within a period (R) does not exceed 2 matches, as specified by the constraint.

  Let's break down each predicate and its purpose:

  - The predicate `period` defines a period by specifying two slots, $X1$ and $X2$, where $X2$ is equal to $X1$ plus a value R. The value of R is obtained from the `max_R` predicate, where each team within a period (R) does not exceed 2 matches, as specified by the constraint.
  - The predicate `home_per_period` counts the number of home matches (`match (T, _, S)`) for a given team T within a period specified by $X1$ and $X2$. The count is assigned to C. The count is obtained by counting S (slots) that fall within the specified period range. The predicate also ensures that $X1$ and $X2$ are valid slots and T is a valid team.
  - The predicate `away_per_period` is similar to `home_per_period`, but it counts the number of away matches (`match (_, T, S)`) for a given team T within the specified period.
  - The predicate `total_per_period` calculates the total number of matches (C) for a given team T within the specified period. It combines the counts from both home matches ($C1$) and away matches ($C2$).
  - The final line ensures that no team T has more than 2 matches ($C > 2$) within any given period defined by slots $S1$ and $S2$. The constraint is applied to all valid combinations of teams, slots, and periods.

```
% Constraint 5 -  each team plays at most 2 games in a period of Rmax time slots
period(X1, X2) :- slot(X1), slot(X2), X2=X1+R,max_R(R).

home_per_period(X1, X2, T, C) :- C=#count{S: match(T, _, S), period(X1, X2), S>=X1, S<=X2},slot(X1),slot(X2),team(T).

away_per_period(X1, X2, T, C) :- C=#count{S: match(_, T, S), period(X1, X2), S>=X1, S<=X2},slot(X1),slot(X2),team(T).

total_per_period(X1, X2, T, C) :- home_per_period(X1, X2, T, C1), away_per_period(X1, X2, T, C2), C=C1+C2.

:- team(T), slot(S1), slot(S2),total_per_period(S1, S2, T, C),C>2.
```

Figure 7: Constraint 5

- Constraint 6: These rules (Figure 8) prevent clashes in match scheduling by ensuring that matches involving the same teams are not scheduled in the same slot and that an appropriate slot difference is maintained, as specified by $m_slot(M)$.

  Let's break down each rule and its purpose:

– The first line checks for a violation where two matches involving the same teams ($H1, A1, H2, A2$) have different slots ($S1$ and $S2$). The rule ensures that $S2$ is strictly greater than $S1$ and that the slot difference is less than M, as defined by `m_slot(M)`.

– The second line is similar to the previous rule but checks for the opposite situation. It verifies whether there is a violation where two matches involving the same teams ($H1, A1, H2, A2$) have different slots ($S1$ and $S2$). The rule ensures that $S1$ is strictly greater than $S2$ and that the slot difference is less than M, as defined by `m_slot(M)`.

```
%Constrain 6 - there are at least m time slots between two games with the same pair of teams.
:- match(H1, A1, S1), match(A2, H2, S2),A1=A2,H1=H2,S1!=S2,S2>S1,(S2-S1) < M, m_slot(M).
:- match(H1, A1, S1), match(A2, H2, S2),A1=A2,H1=H2,S1!=S2,S1>S2,(S1-S2) < M,m_slot(M).
```

Figure 8: Constraint 6

- Optimization Objective: The objective (Figure 9) aims to find the optimal solution that maximizes the overall number of matches scheduled. This can be useful in scenarios where maximizing the number of matches is a desired outcome, such as in sports scheduling or event planning, where increasing the total number of matches can enhance the overall experience or revenue generation.

```
% To maximize total matches count to get optimal solution
#maximize { 1, match(Home,Away,TS) : match(Home,Away,TS) }.
```

Figure 9: optimization

- **Output Visualization:** The given matrix (Figure 10) represents scheduled matches between home teams and away teams. Each row represents a home team, each column represents an away team, and the cell at position (i, j) stores the slot in which the home team at row i plays against the away team at column j. If a match cannot be scheduled, the corresponding cell is marked as "-1". This matrix provides a visual representation of the scheduled slots for matches, with "-1" indicating unscheduled matches.

  The `print_table` function (pythoncodetest.py) takes a table as input and prints it in a visually organized manner. It calculates the maximum width needed for each cell and then iterates over each row and cell, printing the content centered within the allocated space. Vertical bars are used to separate cells, and horizontal lines are printed between rows. The function ensures that the table is printed with proper alignment and visual separation for easy readability.

- ASP Approach 1 vs. Tabu Search Results:

  The graph (Figure 11) compares the matches generated between Tabu Search and Approach 1 of ASP for different leagues in the LZV Cup. Each instance is labeled with an instance input number, and the corresponding values for Tabu Search and Approach 1 results are plotted on the y-axis. The graph shows that, overall, Tabu Search consistently outperforms Approach 1 in generating matches that adhere to the specified constraints and requirements. While Approach 1 closely aligns with Tabu Search values in some instances, there are notable deviations where Tabu Search performs significantly better.

3.4 Approach 2: Penalty for Unscheduled matches.

According to the paper [VGS19],we implemented solution in `unschedule.lp`, penalties are assigned to unscheduled matches in order to find an optimal scheduling solution that minimizes overall penalties while satisfying the specified constraints. The objective is to maximize the utilization of available time slots and ensure that all teams are accommodated within the given time. This approach strikes a balance between minimizing penalties for unscheduled games and efficiently organizing matches. The code (Figure 12) assigns a penalty of 10 to unscheduled matches where one team plays at home but not away, or vice versa. The total penalties are calculated, and the optimization process aims to minimize these penalties. Here is a breakdown of the code:

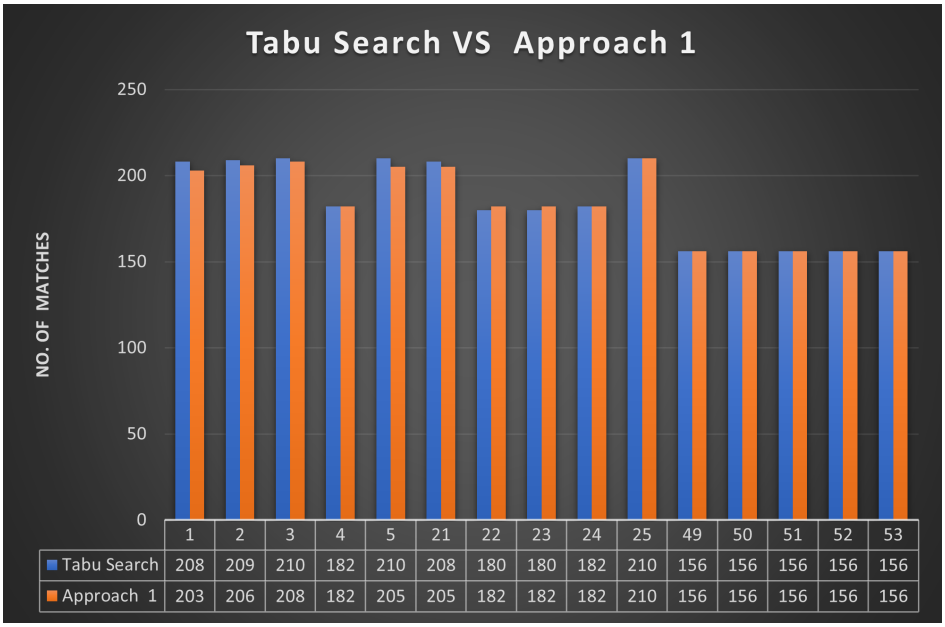
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	-1	39	235	249	25	53	11	67	81	95	109	137	151	165	179
2	228	-1	25	46	60	88	74	102	130	144	158	186	172	200	214
3	37	93	-1	9	51	261	65	79	107	163	177	191	205	219	247
4	17	227	73	-1	59	87	31	101	129	143	157	171	185	199	-1
5	87	213	157	255	-1	3	17	45	73	101	129	143	171	185	199
6	239	163	23	177	65	-1	37	93	79	107	135	225	191	205	211
7	150	136	234	94	80	108	-1	38	52	10	66	164	178	192	206
8	131	33	5	173	145	159	187	-1	47	61	75	89	201	215	229
9	193	235	39	263	137	151	165	109	-1	25	53	67	179	95	207
10	157	31	3	213	269	45	73	171	87	-1	59	185	199	241	255
11	180	54	68	82	194	208	138	152	236	166	-1	96	250	110	264
12	227	3	17	45	-1	31	59	199	157	-1	241	-1	87	73	101
13	31	45	59	269	-1	101	241	73	255	-1	143	213	-1	129	157
14	234	52	80	66	94	136	262	108	164	150	206	248	220	-1	178
15	30	135	16	226	9	114	93	156	121	184	198	-1	86	-1	-1

Figure 10: ASP Output

- The first line assigns a penalty of 10 to a match between teams H and A if there is a match where A is the home team and H is the away team (match(A, H, _)), but there is no match where H is the home team and A is the away team (not match(H, A, _)). This rule penalizes unscheduled matches where one team plays at home but not away or vice versa, violating the round-robin rule.
- The second line is similar to the previous one but assigns a penalty of 10 to a match between teams A and H if there is a match where H is the home team and A is the away team (match(H, A, _)), but there is no match where A is the home team and H is the away team (not match(A, H, _)). This rule penalizes unscheduled matches where one team plays away but not at home.
- The third line calculates the total penalties (Ps) by summing up all the penalties assigned by the penalty predicate. It uses the #sum aggregation to compute the sum of penalties.
- The fourth line defines a minimization statement to find the smallest possible value for the total penalties (Ps). The @2 indicates the priority level of the minimization objective. By minimizing the total penalties, the code aims to find a scheduling solution with the fewest violations of the constraints and penalties assigned to unscheduled matches.
- The last line defines a maximization statement to maximize the total count of matches. It assigns a weight of 1 to each match and uses the @1 priority level. By maximizing the total match count, the code aims to find a scheduling solution with the highest number of scheduled matches, which is considered an optimal solution.

- ASP Approach 1 vs. Approch 2:

  Comparing Approach 2 to Approach 1 (Figure 13) across multiple instances, it can be observed that Approach 2 consistently outperforms Approach 1. Approach 2 yields better results in terms of the number of matches scheduled. This indicates that Approach 2 provides a more optimal and efficient scheduling solution for the given instances.



	1	2	3	4	5	21	22	23	24	25	49	50	51	52	53
Tabu Search	208	209	210	182	210	208	180	180	182	210	156	156	156	156	156
Approach 1	203	206	208	182	205	205	182	182	182	210	156	156	156	156	156

Figure 11: ASP Approach 1 vs. Tabu Search Results

```
%Applying penalty for unschedule match
penalty(H,A, 10) :- team(H),team(A),match(A,H, _), not match(H,A, _).
penalty(A,H, 10) :- team(H),team(A),match(H,A, _), not match(A,H, _).
total_penalties(Ps) :- Ps = #sum{P : penalty(_,_, P)}.
%Minimizing the penalty
#minimize {Ps@2 : total_penalties(Ps)}.
%Minimizing the penalty
% To maximize total matches count to get optimal solution
#maximize { 1@1, match(Home,Away,TS) : match(Home,Away,TS) }.
```

Figure 12: Penalty for unscheduled matches

3.5 Experimentation :

In this section, we conducted two experiments on ASP solution Approach 2: Penalty for Unscheduled matches. as it was giving better results and was generating more number of matches. We provide a detailed explanation of both experiments below.

3.5.1 First Experiment

Here, we conduct experiments with various values of $R_{\max}$ and m parameters in our code and analyze their corresponding outputs. We compare the results using graphs to visualize the performance of different configurations. Our aim is to evaluate and determine the most suitable values of $R_{\max}$ and m for scheduling matches in the LCVZ Cup tournament.

By varying the values of $R_{\max}$ (representing the maximum time gap between consecutive games for a team) and m (representing the number of time slots available for scheduling), we can observe the impact of these parameters on the scheduling outcomes. The experiments provide insights into how different combinations of $R_{\max}$ and m affect the scheduling efficiency, fairness, and overall quality of the tournament schedule.

Using the graphs generated from the experiments, we can assess the trade-offs between different configurations and identify the optimal values of $R_{\max}$ and m. These optimal values can be recommended to the tournament organizers, enabling them to make informed decisions about scheduling matches for the LCVZ Cup. The goal is to suggest the most effective combination of $R_{\max}$ and m that achieves a well-balanced, efficient, and fair tournament schedule.

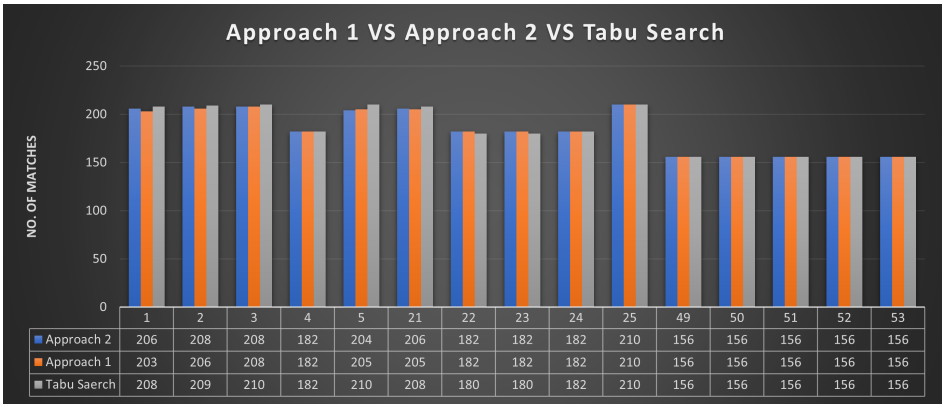


Figure 13: ASP Approach 1 vs. Approach 2

This experiment involves three different types (Type 1, Type 2, and Type 3) with variations in the parameters $R_{\max}$ and m. The graph values were obtained by conducting Experiment for Type 1 (Rmax=4, m=60), Type 2 (Rmax=3, m=40), and Type 3 (Rmax=5, m=80).

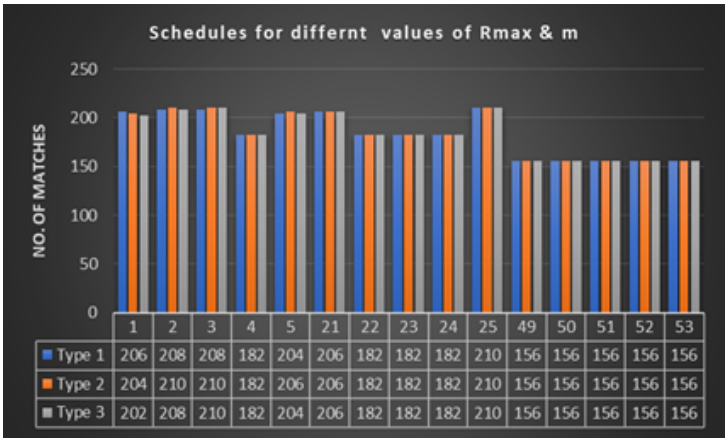


Figure 14: Experiment 1: Type1 vs Type2 vs Type 3

The graph in Figure 14 represents the data that illustrates the performance comparison among three types: Type 1, Type 2, and Type 3. The values represent the number of matches generated for each type. It is evident that Type 2 consistently produces a higher number of matches compared to Type 1 and Type 3. This suggests that Type 2 prioritizes maximizing the number of matches, even if it means sacrificing rest days between matches. Therefore, if the tournament organizer seeks to conduct more matches and is willing to compromise on rest days, Type 2 would be the preferred choice. It offers the opportunity to schedule a greater number of matches, providing more excitement and engagement for the participants and spectators of the tournament.

3.5.2 Second Experiment

The main objective of the LZV Cup is to schedule matches for all teams participating in the tournament. In this experiment , we aim to penalize teams that violate Constraint 5 as shown in Figure 15, which states that each team can play at most two games in the $R_{\max}$ slot. By penalizing teams that exceed this limit, we encourage a fair distribution of matches across different slots. The goal is to minimize the total penalty, which means finding a scheduling solution that results in the fewest violations of Constraint 5. This experiment ensures that the scheduling is balanced and gives each team an equal opportunity to compete in the $R_{\max}$ slot.we have used `aspschedule.lp` file for this experiment.

- Assign a penalty of 10 to a slot pair $(S1, S2)$ if there is a team T for which the total count of matches in that period

```
%Penalty for violating Constarint 5
penalty(S1,S2, 10) :-  team(T), slot(S1), slot(S2),total_per_period(S1, S2, T, C),C>2.
total_penalty(Ps) :- Ps = #sum{P : penalty(_, _, P)}.
%minimizing the penalty
#minimize {Ps@3 : total_penalty(Ps)}.
%Penalty for unschedule match
penalty_un(H,A, 10) :- team(H),team(A),match(A,H, _), not match(H,A, _).
penalty_un(A,H, 10) :- team(H),team(A),match(H,A, _), not match(A,H, _).
total_penalty_un(Ps) :- Ps = #sum{P : penalty_un(_,_, P)}.
%minimizing the penalty
#minimize {Ps@2 : total_penalty_un(Ps)}.
% To maximize total matches count to get optimal solution
#maximize { 1@1, match(Home,Away,TS) : match(Home,Away,TS) }.
```

Figure 15: ASP Code for penalising constraint 5

(between slots $S1$ and $S2$) exceeds 2 ($C > 2$). This rule penalizes violations of Constraint 5, which states that each team can play at most 2 games in the given period.

- Calculate the total penalties (P_s) by summing up all the penalties assigned by the penalty predicate. Use the #sum aggregation to compute the sum of penalties. Then, define a minimization statement to find the smallest possible value for the total penalties (P_s) using the @3 priority level. By minimizing the total penalties, the code aims to find a scheduling solution with the fewest violations of Constraint 5 (Figure 16)and the smallest penalty.

```
------------------------------------------------------------
Team 9 has more than 2 games scheduled within4 time slots.
Violating Match details:
------------------------------------------------------------
3 vs 9 at slot 107
7 vs 9 at slot 108
9 vs 12 at slot 109
```

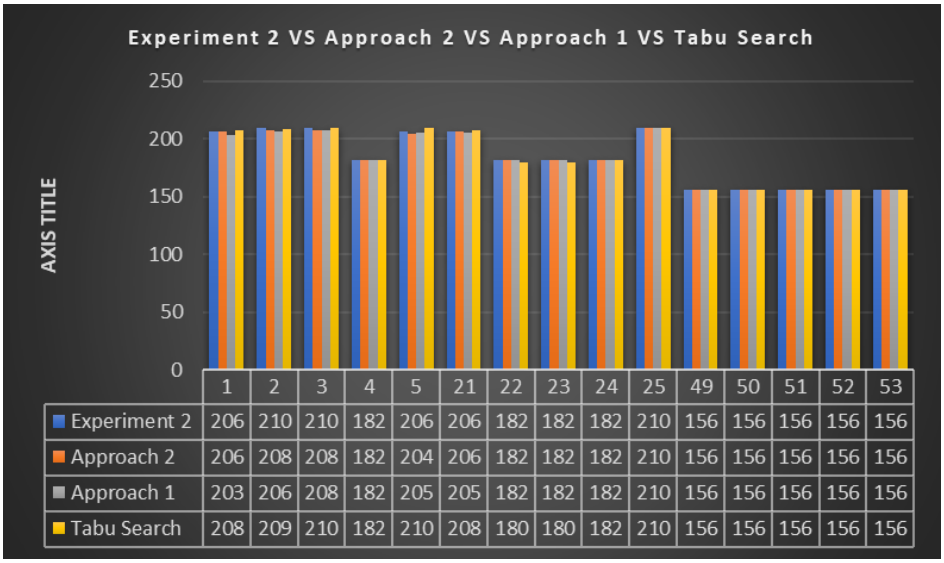
Figure 16: Viaolation of Constraint 5

The graph 17 displays the outcomes of the second experiment, comparing the number of matches generated by Approach 2, Approach 1, and Tabu Search with Experiment 2. Notably, Experiment 2 generates a significantly higher number of matches compared to the other approaches. However, upon closer examination, it becomes apparent that there are instances where Constraint 5, which limits the number of matches within a specified rest day period, is violated in Experiment 2. Despite this violation, if the organizer's primary objective is to schedule all matches without prioritizing rest days, then Experiment 2 proves to be a more suitable option. It enables the scheduling of all matches, albeit with potential compromises on rest days.

3.6 Evaluation

We evaluate the output results of our ASP scheduling algorithm using two complementary approaches. The first approach involves implementing Python code to check each constraint individually, ensuring that the scheduled games comply with all the specified requirements. This meticulous evaluation helps us identify any violations of constraints in our output.

In the second approach, we utilize the calendar output provided in the paper and convert it into our knowledge base. By running our own constraints on this calendar-based output, we can assess its compatibility with our specific criteria. By combining these evaluation methods, we can thoroughly analyze the quality and feasibility of our scheduling solution, ensuring it meets the desired constraints and produces optimal results.



	1	2	3	4	5	21	22	23	24	25	49	50	51	52	53
Experiment 2	206	210	210	182	206	206	182	182	182	210	156	156	156	156	156
Approach 2	206	208	208	182	204	206	182	182	182	210	156	156	156	156	156
Approach 1	203	206	208	182	205	205	182	182	182	210	156	156	156	156	156
Tabu Search	208	209	210	182	210	208	180	180	182	210	156	156	156	156	156

Figure 17: Experiment 2 vs Others

3.6.1 Evaluation 1:

We have developed a Python script, named `approch1.py`, to evaluate each constraint separately in detail. This code assesses whether each constraint is satisfied or violated by the output of our scheduling algorithm.

The code systematically checks each constraint and examines if it is met or not. If any constraint is found to be violated, the code generates a message indicating which specific constraint has been violated. On the other hand, if all constraints are satisfied, the code displays a message confirming that all constraints have been successfully met.

By implementing this code, we can perform a comprehensive assessment of our scheduling output, ensuring it adheres to the defined constraints. This allows us to identify and address any discrepancies or issues in the scheduling solution, enabling us to refine and improve the quality of our results.

The python script is tested for different scenarios before testing the ASP solution on it. We did boundary value testing,Negative scenario testing, and positive scenario testing.

3.6.2 Evaluation 2:

To validate that our solutions generated by our ASP code is valid and to check that our solution is bound to all 6 constraints we did a reverse engineering. We converted matches generated by Tabu search into availability by a script `calander.py` the script converts the row and columns into [team1(home team),team2(away team),slot] and we compare each team and its availability from respective input.txt file and generate availability for those slots which were given in tabu search solution. only those availability is fed them to our ASP solution and we got similar result that was generated by tabu search.(Figure 18)

To execute this, we follow these steps:

- Select the files from the output of Tabu search in the "CalandersTabu" folder that we want to test our code on. Place these selected files in a separate folder or directory.
- Run our Python script `calander.py` to generate the availability. Make sure to have all the necessary dependencies and libraries installed for successful execution of the script.
- Once the `calander.py` script has completed its execution, run the `calender_match_generator.py` script. This script generates the matches and output matrix based on the previously generated availability.

Following these steps allows us to test our code using the selected files from the Tabu search output and generate the desired matches and output matrix.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	-1	185	87	241	45	59	269	129	101	143	213	73	171	64	92
2	66	-1	234	178	52	80	206	164	38	10	94	150	262	248	227
3	206	108	-1	52	80	164	38	10	94	150	262	248	227	241	45
4	108	87	185	-1	66	206	80	38	164	94	10	227	150	45	248
5	234	269	143	262	-1	108	164	206	10	38	150	94	248	227	185
6	178	241	66	269	213	-1	234	94	150	262	38	10	185	87	101
7	52	45	178	143	87	129	-1	108	66	248	227	262	10	94	150
8	262	59	101	234	24	248	185	-1	178	66	52	80	87	150	241
9	248	143	269	59	73	52	241	45	-1	108	234	206	80	262	87
10	80	129	59	213	178	45	101	227	185	-1	206	234	52	164	269
11	24	171	129	101	59	218	73	269	64	87	-1	108	66	178	80
12	164	213	24	129	171	143	59	218	92	64	185	-1	178	66	38
13	38	101	73	218	129	64	213	204	190	241	164	45	-1	108	206
14	218	24	171	204	101	190	8	213	129	92	78	269	234	-1	52
15	204	64	218	78	8	246	50	106	176	190	162	198	94	156	-1

Figure 18: Second Approach Output

3.7 Conclusion:

We have run all our ASP solution on parallel mode enabled with 8 cores on a windows 11 OS with an Intel 11th Gen Intel(R) Core(TM)i7 processor running at 3.0 GHz and provided with 16GB of ram and were run for 1200 seconds each.

All our ASP solutions were implemented with a time break of 1200 seconds every time this time break got executed which means the solution we are providing is only one of the optimized solution and not the perfect optimization and there was still a chance for getting a much better optimized solution if we had increased out time limit and if the code was run on better CPU with more threads and better computational power.

We conclude that we were very close to our solution compared to the Tabu search results implemented in [VGS19].

4 SWI-Prolog.

Prolog, a logic programming language, is commonly used for tournament scheduling due to its rule-based logic, backtracking capabilities, symbolic manipulation, easy prototyping, integration with databases, and proven track record. Its rule-based programming allows for expressing complex scheduling constraints, while backtracking enables exploration of scheduling options.

We implemented SWI-Prolog solution for scheduling LZV Cup. In the given code, constraints are rules or conditions that must be strictly satisfied in order to find a valid solution. In this context, we have implemented only 4 constraints referring to the paper [VGS19] and the constraints are as follows:

- Constraint 1: Each team plays a home game against each other team at most once. Violation of this constraint would result in a team playing multiple home games against the same opponent.
- Constraint 2: Home team availability is respected. This constraint ensures that a team can only play a home game if they are available.
- Constraint 3: Away team unavailability is respected. This constraint guarantees that a team does not play an away game if they are unavailable.
- Constraint 4: Each team plays at most one game per time slot. This constraint ensures that no team is scheduled to play multiple games simultaneously.

4.1 Knowledgebase Creation:

The data generated for the Prolog scheduling problem is similar to the data generated for the ASP scheduling problem. Additionally, there are two predicates: "team," which represents each team, and "slot," which represents each time slot. These predicates provide the necessary information about the teams and time slots for the scheduling process. The generated instances for the Prolog scheduling problem are stored in the "instances_prolog" folder. This organized structure allows for easy access to the generated instances and ensures that the necessary data is available for scheduling using Prolog.

4.2 Required Predicates:

These predicates as shown in Figure 19 provides a basic framework to determine whether a team plays at home, away, or is forbidden to play in a given time slot, based on their availability status, considering the home team should not be equal to the away team.

- The `home/2` predicate checks if a team (`Team`) plays at home in a specific time slot (`Slot`). It does this by checking the availability of the team in that slot using the `availability/3` predicate. If the availability is set to '1', it means the team plays at home.
- The `away/2` predicate checks if a team (`Team`) plays away in a specific time slot (`Slot`). Similar to the `home/2` predicate, it uses the `availability/3` predicate to check the availability of the team. If the availability is set to '0', it means the team plays away.
- The `forbidden/2` predicate checks if a team (`Team`) is forbidden to play in a specific time slot (`Slot`). Once again, it relies on the `availability/3` predicate and checks if the availability is set to '2'. If so, it indicates that the team is forbidden to play.
- The `condition1/2` predicate is not directly related to the availability, but it appears to be a helper predicate to check if the home team (`Home`) is not equal to the away team (`Away`). It uses the $\backslash =$ operator, which means "not equal." If the home and away teams are different, the condition evaluates to true.

4.3 Designing Prolog Code:

- The `constraint2/2` predicate enforces the constraint that the home team (`H`) must be available (`home(H, S)`) in a specific time slot (`S`) and must not be forbidden to play (`not(forbidden(H, S))`). In other words, it ensures that the home team is both available and not forbidden to play. (Figure 20)
- The `constraint3/2` predicate enforces the constraint that the away team (`A`) must not be forbidden to play (`not(forbidden(A, S))`) in a specific time slot (`S`). It ensures that the away team is not forbidden to play. (Figure 20)
- The `match/3` predicate generates all possible matches by considering valid teams, distinct home and away teams, and valid time slots. It ensures that the home team is available and not forbidden to play in a given time slot, while the away team is not forbidden to play. By combining these predicates, `match/3` produces valid match combinations that adhere to the availability constraints for both the home and away teams. (Figure 20)

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Predicates for LZV cup tournament
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This predicate gives  whether a team plays in home  or not:
home(Team,Slot):-availability(Team,Slot,'1').
% This predicate gives  whether a team plays  away  or not:
away(Team,Slot):-  availability(Team,Slot,'0').
% This predicate gives  whether a teams is forbidden to play or not:
forbidden(Team,Slot):-availability(Team,Slot,'2').
%To check home team is not equal toaway team
condition1(Home, Away) :-
    Home \= Away.
```

Figure 19: Prolog Predicates.

```
%Constrain 2 -home team availability Hi (i ∈ T ) is respected
constarint2(H,S):- home(H,S),not(forbidden(H,S)).
%constrain 3 -  away team unavailability Ai (i ∈ T ) is respected
constarint3(A,S):- not(forbidden(A,S)).
%To generate all match with home and away availability
match(H,A,S):-team(H),team(A),condition1(H, A),slot(S),
constarint2(H,S),
constarint3(A,S).
```

Figure 20: Constraints 2&3.

The `filter_matches/4` predicate filters a list of matches by checking various conditions to ensure that each team plays a home game against each other team at most once and that each team plays at most one game per time slot. It maintains a processed list of matches and a helper list, accumulating the matches that satisfy the constraints. The `constraint_1_4/2` predicate serves as a wrapper, utilizing `filter_matches/4` to enforce both constraints and provide the filtered list of matches that meet the requirements. (Figure 21)

The `constraint_2_3/1` predicate creates a list of all possible matches by combining the `match/3` predicate with the constraints `constarint2/2` and `constarint3/2`, using `findall/3` to gather and store the matches in the `Schedule` list. The `all_match/1` predicate generates all valid matches by invoking `constraint_2_3/1` to obtain the list of possible matches (`Slots`), applies the `constraint_1_4/2` predicate to filter and enforce constraints, stores the filtered matches in the `FinalSchedule` list, and writes the matches to a file using `write_matches/2`. The process completes by closing the file with `close/1`. In summary, `all_match/1` combines the necessary predicates to generate and store the valid matches, while `constraint_2_3/1` and `constraint_1_4/2` handle the specific filtering and constraint enforcement. (Figure 22)

4.4 Output Visualization:

We have developed a Python code (`prolog.py`) to convert the list of matches generated by running the `all_match` predicate into a matrix format. In this format, each row represents a home team, each column represents an away team, and the cell at

```
% Constraint 1 -each team plays a home game against each other team at most once
% Constrain 4 - each team plays at most one game per time slot
filter_matches([], _, Helper, Helper).
filter_matches([matches(H,A,S)|Tail], Processed, Helper, Filtered) :-
    \+ member(matches(H,A,S), Processed),
    \+member(matches(H,_,S), Processed),
    \+member(matches(_,A,S), Processed),
    \+member(matches(_,H,S), Processed),
    \+member(matches(A,_,S), Processed),
    \+member(matches(H,A,_), Processed), % This deals with Constraint 1
    filter_matches(Tail, [matches(H,A,S)|Processed], [matches(H,A,S)|Helper], Filtered).
filter_matches([_|Tail], Processed, Helper, Filtered) :-
    filter_matches(Tail, Processed, Helper, Filtered).

constraint_1_4(Matches, Filtered) :-
    filter_matches(Matches, [], [], Filtered).
```

Figure 21: Constraints 1&4.

```
%To create list of all matches
constraint_2_3( Schedule) :-
    findall(matches(H,A,S), match(H,A, S), Schedule).
%To generate all matches
all_match(FinalSchedule) :-constraint_2_3(Slots),
constraint_1_4(Slots, FinalSchedule),
    open('Prolog_output.txt', write, Stream),
    write_matches(Stream, FinalSchedule),
    close(Stream).
```

Figure 22: All Matches.

position (i, j) stores the slot in which the home team i plays against the away team j. If a match could not be scheduled, the cell is filled with a "-1" to indicate the unavailability. This conversion (Figure 23) allows for a more structured representation of the match schedule, facilitating further analysis and processing of the data.

4.5 Conclusion for SWI-Prolog Solution :

As we have only implemented 4 constraints in SWI-Prolog we are not able to do evaluation of the output with ASP solution or with Tabu Search. For 1st league `input1.txt` we were able to generate 206 matches and our solution did not violate any of the implemented constrains.The output generated was tested with the help of python using`prolog.py` file.Our further work will be implementing the remaining 2 constraints related to constraints 5 and 6.

5 How to run the code:

Environment Configuration Specification Visual Studio with Python, Swipl & Clingo

- Step 1: Login to GitHub and clone our project.`https://github.com/RMIT-COSC2780-IDM23/project-ddjs.git`
- Step 2: Do git fetch and pull in the IDE, Visual Studio.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	-1	25	39	235	11	53	67	81	95	109	137	151	165	179	193
2	228	-1	46	60	74	88	102	130	144	158	172	186	200	214	242
3	23	37	-1	9	51	65	79	93	107	163	177	191	205	219	233
4	17	31	59	-1	73	87	101	129	143	157	171	185	199	213	227
5	45	3	17	255	-1	59	87	101	129	143	157	171	185	199	213
6	37	23	211	79	93	-1	107	135	163	177	191	205	218	225	232
7	38	52	66	10	80	94	-1	108	136	150	164	192	178	206	234
8	33	5	47	61	75	89	103	-1	131	145	159	187	173	201	215
9	207	39	25	249	53	67	81	109	-1	151	165	137	179	193	235
10	31	45	3	241	269	73	59	87	101	-1	185	199	171	255	-1
11	54	68	82	96	110	138	152	166	180	194	-1	208	236	250	264
12	59	17	31	45	227	3	73	143	87	213	101	-1	129	157	241
13	73	59	45	269	-1	31	143	213	157	227	87	255	-1	101	-1
14	52	38	80	66	10	108	220	94	150	164	136	234	192	-1	178
15	16	30	86	93	9	114	121	156	135	184	198	226	212	-1	-1

Figure 23: SWI-Prolog Output

- Step 3: Run the ASP solution by following the step-by-step instructions below and verify the output.
- Step 4: Run the Prolog solution by following the step-by-step instructions below and verify the output.

ASP Solution of LZV Cup: Files Descriptions & Execution Order

1. File Name: `datagen_asp.py`
   - Description: To convert input instance from paper into knowledge base.
   - How to Run: On terminal, run the following command: `python .\datagen_asp.py`
   - Output folder Name: `instances_asp`
   - Dependencies: Input folder data from paper.
2. File Name: `schedule.lp`
   - Description: This file consists of code for Approach 1.
3. File Name: `approach1.py`
   - Description: This file consists of Python code to run Approach 1 integrated with `schedule.lp`.
   - How to Run: On terminal, run the following command: `python .\approach1.py`
   - Output folder Name: `approach1`
   - Dependencies: `schedule.lp`, `instances_asp`
4. File Name: `unschedule.lp`
   - Description: This file consists of code for Approach 2.
5. File Name: `approach2.py`
   - Description: This file consists of Python code to run Approach 2 integrated with `unschedule.lp`.
   - How to Run: On terminal, run the following command: `python .\approach2.py`
   - Output folder Name: `approach2`
   - Dependencies: `unschedule.lp`, `instances_asp`
6. File Name: `experiment1_Type1.py`

- Description: To run type 1 experiment-1 on Approach 2.
- How to Run: On terminal, run the following command: `python .\experiment1_Type1.py`
- Output folder Name: `experiment1_Type1`
- Dependencies: `unschedule.lp`, change values of `max_R=4`, `m_slot=60`

7. File Name: `experiment1_Type2.py`
   - Description: To do type 2 experiment-1 on Approach 2.
   - How to Run: On terminal, run the following command: `python .\experiment1_Type2.py`
   - Output folder Name: `experiment1_Type2`
   - Dependencies: `unschedule.lp`, change values of `max_R=3`, `m_slot=40`
8. File Name: `experiment1_Type3.py`
   - Description: To do type 3 experiment-1 on Approach 2.
   - How to Run: On terminal, run the following command: `python .\experiment1_Type3.py`
   - Output folder Name: `experiment1_Type3`
   - Dependencies: `unschedule.lp`, change values of `max_R=5`, `m_slot=80`
9. File Name: `aspschedule.lp`
   - Description: This file consists of code for Approach 2-Experiment2.
10. File Name: `experiment2.py`
    - Description: To do experiment-2 on Approach 2 integrated with `aspschedule.lp`.
    - How to Run: On terminal, run the following command: `python .\experiment2.py`
    - Output folder Name: `experiment2`
    - Dependencies: `aspschedule.lp`, `unschedule.lp`

Prolog Solution of LZV Cup: Files Descriptions & Execution Order

A. File Name: `datagen_prolog.py`
   - Description: To convert input instance from the paper into a knowledge base.
   - How to Run: On the terminal, run the following command: `Python .\datagen_prolog.py`
   - Output Folder Name: `instances_prolog`
   - Dependencies: Input folder data from the paper.

B. File Name: `datagen.py`
   - Description: To create instances folder to generate output matrix.
   - How to Run: On the terminal, run the following command: `Python .\datagen.py`
   - Output Folder Name: `instances`

C. File Name: `schedule.pl`
   - Description: This file consists of code for Prolog solution for LZV cup.
   - How to Run:
     1. On the terminal, run the following command:
        `swipl -f .\instances_prolog\instance1.lp .\schedule.pl`
     2. Note: `swipl -f .\instances_prolog\instance[i].lp`, here, please change `i` to 1, 2, 3, and so on to test output of different instances.
     3. In the query, pass: `all_match(X).`

4. Exit the query by pressing a period or closing the terminal.

– Output Folder Name: `prologoutput.txt`

– Dependencies: 1. `datagen_prolog.py` 2. `datagen.py` 3.

D. File Name: `prolog.py`

– Description: To generate the output matrix of Prolog.

– How to Run: On the terminal, run the following command: `Python .\prolog.py`

– Output Folder: N/A, Console

– Dependencies: 1. `datagen_prolog.py` 2. `datagen.py` 3. `schedule.pl`

Evaluation of ASP Solution:

After executing the above steps, execute the following steps to assess whether each constraint is satisfied or violated.

Evaluation 1:

1. File name: `approach1.py`
   - How to Run: On terminal, run the following command: `Python .\approach1.py`
   - Output folder Name: `approach1`
2. File name: `approach2.py`
   - How to Run: On terminal, run the following command: `Python .\approach2.py`
   - Output folder Name: `approach2`
3. File name: `experiment2.py`
   - How to Run: On terminal, run the following command: `Python .\experiment2.py`
   - Output folder Name: `approach3`

Evaluation 2:

1. From folder "CalandersTabu", select `calendar_1_0.dat` or any other input file that you want to test and place it in the `calanderTestInput` folder.
2. File name: `calendar.py`
   - How to Run: On terminal, run the following command: `Python .\calendar.py`
   - Output folder Name: `calanderInstance`
3. File name: `calendar_match_generator.py`
   - How to Run: On terminal, run the following command: `Python .\calendar_match_generator.py`
   - Output folder Name: `output_calendar`

Please do reach to us if you face any issues while executing any of the step.

6 References

[KL09] Sigrid Knust and Daniel Lücking. "Minimizing costs in round robin tournaments with place constraints". In: Computers & Operations Research 36.11 (2009), pp. 2937–2943.

[Knu10] Sigrid Knust. "Scheduling non-professional table-tennis leagues". In: European Journal of Operational Research 200.2 (2010), pp. 358–367.

[NGK14] Kimmo Nurmi, Dries Goossens, and Jari Kyngäs. "Scheduling a triple round robin tournament with minitournaments for the Finnish national youth ice hockey league". In: Journal of the Operational Research Society 65.11 (2014), pp. 1770–1779.

[Kyn+17] Jari Kyngäs et al. "Scheduling the Australian football league". In: Journal of the Operational Research Society 68.8 (2017), pp. 973–982.

[VGS19] David Van Bulck, Dries R Goossens, and Frits CR Spieksma. "Scheduling a non-professional indoor football league: a tabu search based approach". In: Annals of Operations Research 275.2 (2019), pp. 715–730.

[CA20] Burak Çavdaroğlu and Tankut Atan. "Determining matchdays in sports league schedules to minimize rest differences". In: Operations Research Letters 48.3 (2020), pp. 209–216.