⚾ 📅 📄

Little League Competition Schedule Report

IND ENG 162: Linear Programming & Network Flows

By: Angelina Bai, Sera Göksal, Yang Lu, James Shi

---

1. EXECUTIVE SUMMARY

In this report, we present the modeling and optimization of a Little League baseball match schedule through an Optimization Integer Program, tailored to meet specific criteria and constraints, stated by the League's manager. We have constructed integer programs & generated schedule solutions that adhere to the requirements set forth, ensuring all scheduled games fit within the constraints highlighted in the initial request. For the manager's first ask we obtained a feasible optimal schedule. When we relaxed the condition of having precisely four games every Saturday to allow for up to four, it resulted in a different, yet still feasible, schedule. However, the manager's additional request for a game on every possible weekday introduces a complexity that the model cannot accommodate, leading to the conclusion that under such restrictive conditions, no perfect scheduling solution is possible. The detailed process and the resulting schedule variations, compliant with the varying constraints, are meticulously outlined in the subsequent sections of this report.

---

2. BACKGROUND

There are 9 teams in total, each uniquely identified by a number ranging from 1 to 9. The objective is for each team to play against every other team twice over the baseball season. The competition schedule lasts three months from March to May, considering holidays and vacations. It begins on March 13 and ends on May 31. Dates are organized in blocks of four, with the exception of the last block, including May 30 and 31. A total of 72 competitions (36 pairs) need to be held.

And following constraints need to be satisfied (for part 1).

- To ensure a balanced schedule, there is a limit of one game per weekday.
- There is at most one game per weekday
- Competitions are scheduled from Monday through Thursday every week.
- No team should participate more than once within any four-day period in a given week.
- A weekly Saturday competition involves four simultaneous matches.

- Teams that participated in a Monday-to-Thursday competition can also partake in the subsequent Saturday's competition.

For part 2, we would like to replaces the strict requirement of having exactly 4 games every Saturday to have at most 4 games. Lastly, we would like to arrange the schedule so that games will be played on every possible weekday in the schedule. To solve the problem, we will set up three integer programs according to different constraints and excels provided.

---

3. MODELING

In this section, we will dive into how we developed our integer programming model in AMPL to create an optimal and feasible schedule for the Little League competitions. We will begin with defining all the requirements for formulation and modeling for part 1 and then additionally discuss the changes made to the AMPL model for parts 2 and 3.

Sorted Excel

Given the original Excel data, we first deleted all dates where there would be no games at all (holidays), and we ended up with 50 days where 8 were Saturdays. We sorted the data in order to assign each game day an integer order so all days would be visualized consecutively. By doing this, we can formulate and solve the integer program for this scheduling feasibility problem without having to directly import the actual Excel file into AMPL.

Season Duration & Holiday Constraints
The letter from the Little League's manager mentions that the Excel file attached to the letter provides the calendar schedule, spanning three months from March to May. The schedule considers holidays and vacations, commencing on March 13 and concluding on May 31. The competition dates are organized in blocks of four, except for the last one, which includes May 30 and 31, as all competitions must conclude by May 31. Below is how we have incorporated these considerations into our model.
In our initial IP formulation (before conversions into AMPL and sorting our Excel), we had also determined the following constraints to factor into our model. As explained below, we sorted and edited the original Excel file to exclude holidays and Sundays as well as abide by the season start and end dates, to be constrained by requirements such as the constraints

numbered 6 and 7 below.

6. **Season Duration Constraint:** The schedule must start on March 13 and end on May 31.

7. **Specific Dates Constraint:** Account for the specific calendar schedule, holidays, and vacations as per the attached Excel file.

<u>Dates and Days W/O Holidays</u>

A	B	C	D	
DAY	DATE		WEEKDAY	DAT
Monday	13-Mar		Monday	13
Tuesday	14-Mar		Tuesday	14
Wednesday	15-Mar		Wednesday	15
Thursday	16-Mar		Thursday	16
Saturday	18-Mar		Monday	20
Monday	20-Mar		Tuesday	21
Tuesday	21-Mar		Wednesday	22
Wednesday	22-Mar		Thursday	23
Thursday	23-Mar		Monday	27
Saturday	25-Mar		Tuesday	28
Monday	27-Mar		Wednesday	29
Tuesday	28-Mar		Thursday	30
Wednesday	29-Mar		Monday	3
Thursday	30-Mar		Tuesday	4
Saturday	1-Apr		Wednesday	5
Monday	3-Apr		Thursday	6
Tuesday	4-Apr		Monday	17
Wednesday	5-Apr		Tuesday	18
Thursday	6-Apr		Wednesday	19
Monday	17-Apr		Thursday	20
Tuesday	18-Apr		Monday	24
Wednesday	19-Apr		Tuesday	25
Thursday	20-Apr		Wednesday	26
Saturday	22-Apr		Thursday	27
Monday	24-Apr		Monday	1-
Tuesday	25-Apr		Tuesday	2-
Wednesday	26-Apr		Wednesday	3-
Thursday	27-Apr		Thursday	4-
Saturday	29-Apr		Monday	8-
Monday	1-May		Tuesday	9-
Tuesday	2-May		Wednesday	10-
Wednesday	3-May		Thursday	11-
Thursday	4-May		Monday	15-
Saturday	6-May		Tuesday	16-
Monday	8-May		Wednesday	17-
Tuesday	9-May		Thursday	18-
Wednesday	10-May		Monday	22-
Thursday	11-May		Tuesday	23-
Saturday	13-May		Wednesday	24-
Monday	15-May		Thursday	25-
Tuesday	16-May		Tuesday	30-
Wednesday	17-May		Wednesday	31-
Thursday	18-May			
Saturday	20-May			
Monday	22-May			

<u>Explanation</u>: In order to overcome the issue/errors with exporting the whole original excel spreadsheet into AMPL, we restructured/sorted our data in Excel to model in AMPL. This sorted table has all the holidays and days that no game is

<u>Part 1 Verified Based on Dates</u>

Table 1

Day		Pt2	DAY	DATE
1	6v8		Monday	13-Mar
2	7v9	2v9	Tuesday	14-Mar
3	2v3	5v8	Wednesday	15-Mar
4	1v5	1v7	Thursday	16-Mar
5	3v5, 4v9, 2v7, 3v9	1v4, 2v4, 1v6, 3v8	Saturday	18-Mar
6	5v9	2v9	Monday	20-Mar
7	3v4	7v8	Tuesday	21-Mar
8	1v2	5v6	Wednesday	22-Mar
9	6v7	3v4	Thursday	23-Mar
10	2v7, 1v6, 4v7, 5v9	1v8, 1v2, 2v4, 7v9	Saturday	25-Mar
11	6v9	4v9	Monday	27-Mar
12	4v5	2v8	Tuesday	28-Mar
13	7v8	1v5	Wednesday	29-Mar
14	1v2	3v6	Thursday	30-Mar
15	3v9, 1v4, 3v6, 2v8	3v8, 4v9, 2v6, 5v9	Saturday	1-Apr
16	2v4	8v9	Monday	3-Apr
17	1v3	2v3	Tuesday	4-Apr
18		1v6	Wednesday	5-Apr
19	5v6	4v5	Thursday	6-Apr
20	1v4	1v7	Monday	17-Apr
21	7v8	4v6	Tuesday	18-Apr
22	2v6	3v5	Wednesday	19-Apr
23	3v5	8v9	Thursday	20-Apr
24	4v9, 2v6, 3v8, 6v9	1v8, 2v6, 5v7, 2v8	Saturday	22-Apr
25		2v3	Monday	24-Apr
26	5v7	6v8	Tuesday	25-Apr
27	3v6	1v5	Wednesday	26-Apr
28	1v8	4v7	Thursday	27-Apr
29	2v3, 4v7, 1v8, 8v9	1v4, 1v3, 5v6, 6v9	Saturday	29-Apr
30	2v8	1v9	Monday	1-May
31	5v7	2v7	Tuesday	2-May
32	4v6	4v5	Wednesday	3-May
33	1v9	6v8	Thursday	4-May
34	2v5, 7v9, 4v8, 1v9	2v7, 3v9, 4v8, 5v7	Saturday	6-May
35	6v7	3v5	Monday	8-May
36	2v9	4v6	Tuesday	9-May
37	1v3	1v2	Wednesday	10-May
38	4v8	7v8	Thursday	11-May
39	1v5, 3v7, 6v8, 8v9	1v9, 2v5, 6v7, 3v9	Saturday	13-May
40	5v8	6v7	Monday	15-May
41	1v6	5v8	Tuesday	16-May
42	2v4	1v3	Wednesday	17-May
43	3v7		Thursday	18-May
44	3v4, 4v5, 5v8, 2v9	4v7, 3v4, 3v7, 5v9	Saturday	20-May
45	2v5	3v7	Monday	22-May
46	1v7	4v8	Tuesday	23-May
47	4v6	2v5	Wednesday	24-May
48	3v8	6v9	Thursday	25-May
49	5v6	7v9	Tuesday	30-May
50	1v7	3v6	Wednesday	31-May

<u>Explanation</u>: This table helped us to verify if the generation of games is following the constraints. We also used this table to check if we missed any necessary constraints. One constraint we added after visualizing the outputs on this table is the *Non_Overlap_Constraint3*, because we observed that there were indeed 2 "Team 1 vs Team 5" games on Day 5, for example.

<u>Part 1 Verified Based on Teams</u>

Team v Team	Day1	Day2		Day	#ofGame(pt2)	#ofGame(pt1)
1v2	37	10		1		1
1v3	29	42		2	1	1
1v4	5	29		3	1	1
1v5	13	27		4	1	1
1v6	5	18		5	4	4
1v7	4	20		6	1	1
1v8	10	24		7	1	1
1v9	39	30		8	1	1
2v3	17	25		9	1	1
2v4	5	10		10	4	4
2v5	47	39		11	1	1
2v6	24	15		12	1	1
2v7	31	34		13	1	1
2v8	12	24		14	1	1
2v9	6	2		15	4	4
3v4	9	44		16	1	1
3v5	35	22		17	1	1
3v6	14	50		18	1	
3v7	45	44		19	1	1
3v8	15	5		20	1	1
3v9	34	39		21	1	1
4v5	32	19		22	1	1
4v6	36	21		23	1	1
4v7	28	44		24	4	4
4v8	34	46		25	1	
4v9	11	15		26	1	1
5v6	29	8		27	1	1
5v7	24	34		28	1	1
5v8	41	3		29	4	4
5v9	15	44		30	1	1
6v7	39	40		31	1	1
6v8	26	33		32	1	1
6v9	29	48		33	1	1
7v8	7	38		34	4	4
7v9	49	10		35	1	1
8v9	16	23		36	1	1
				37	1	1
				38	1	1
				39	4	4
				40	1	1
				41	1	1
				42	1	1
				43		1
				44	4	4
				45	1	1
				46	1	1
				47	1	1
				48	1	1
				49	1	1
				50	1	1
					72	72

<u>Explanation</u>: Similar to the table on the left, this table also serves for us to verify the constraints. We used this table to check if each pair of games is abiding by the constraints. We also used this table to check if the total number of games is 72.

played (Fridays and Sundays) excluded so we don't have to write a constraint to avoid games on those dates. It also assigns each date a number from 1 to 50 so we can make sure the game starts on day 1 and ends on day 50. In the Excel we use, we have 2 columns with Saturdays and 2 columns without Saturdays to visualize the number of weekdays and Saturdays.		

Formulating the Integer Program & Converting into AMPL Code

Our objective is to satisfy all constraints, which makes this a feasibility problem, meaning that we don't necessarily need an explicit objective function. In order to formulate the integer program, we defined 2 essential components: the decision variables and the constraints.

Part 1

```
set TEAMS := 1..9;
set DAYS := 1..50;
set SATURDAYS := {5, 10, 15, 24, 29, 34, 39, 44};

var x{TEAMS, TEAMS, DAYS} binary;

subject to

Schedule_Constraint {i in TEAMS, j in TEAMS: i != j}:
  sum {t in DAYS} (x[i, j,t] + x[j, i, t]) = 2;

Total_Output_Constraint:
  sum {i in TEAMS, j in TEAMS, t in DAYS} x[i, j, t] = 72;

One_Game_Per_Weekday_Constraint {t in DAYS diff SATURDAYS}:
  sum {i in TEAMS, j in TEAMS: i != j} x[i,j,t] <= 1;

Four_Day_Period_Constraint {i in TEAMS, d in {1, 6, 11, 16, 20, 25, 30, 35, 40, 45, 49}}:
  sum {j in TEAMS, t in d..min(d+3, 50): j != i} (x[i,j,t] + x[j,i,t]) <= 1;

Saturday_Competition_Constraint1 {t in SATURDAYS}:
  sum {i in TEAMS, j in TEAMS: i != j} x[j,i,t] = 4;

Non_Overlap_Constraint1 {i in TEAMS, j in TEAMS, k in TEAMS, t in DAYS: i != j && j != k && i != k}:
  x[i,k,t] + x[j,k,t] <= 1;

Non_Overlap_Constraint2 {i in TEAMS, j in TEAMS, k in TEAMS, t in DAYS: i != j && j != k && i != k}:
  x[k,i,t] + x[k,j,t] <= 1;

Non_Overlap_Constraint3 {i in TEAMS, j in TEAMS, t in DAYS: i != j}:
  x[i,j,t] + x[j,i,t] <= 1;
solve;
```

<table>
<tr><th colspan="2">Decision Variables</th></tr>
<tr><td colspan="2">Ask: We are asked by the Little League coach to abide by the constraint “Each team must play against every other team exactly twice.”</td></tr>
<tr><th>Formulation</th><th>Implementation Into AMPL</th></tr>
<tr><td>

X_{ijt}

We define X_{ijt} as our binary decision variable which will equal 1 if **team i** plays against **team j** on **day t**, and 0 otherwise.

Teams i: {1, 2, 3, 4, 5, 6, 7, 8, 9}
Teams j: {1, 2, 3, 4, 5, 6, 7, 8, 9}
Days t: {1, 2, 3, 4, 5, 6} where 1=Monday, 2=Tuesday, 3=Wednesday, 4=Thursday, 5=Friday, 6=Saturday

(Since we have excluded Sundays

</td><td>

```
set TEAMS := 1..9;
set DAYS := 1..50;
set SATURDAYS := {5, 10, 15, 24, 29, 34, 39, 44};

var x{TEAMS, TEAMS, DAYS} binary;
```

set TEAMS := 1..9;
set DAYS := 1..50;
set SATURDAYS := {5, 10, 15, 24, 29, 34, 39, 44};
var x{TEAMS, TEAMS, DAYS} binary;

Explanation: From the letter, we knew there are 9 teams in total, and after we sorted the Excel sheet we assigned 50 game days with consecutive integer order. The set SATURDAYS is a subset of DAYS

</td></tr>
</table>

from our Excel as no teams can play on Sundays, we will not be having 7 as a value in the set of t days)	including specific days {5, 10, 15, 24, 29, 34, 39, 44}, which are designated as the days when Saturday matches occur. We created this to utilize when we're formulating our constraints in AMPL. We set binary variable X_{ijt} as the match between team i against team j on day t. If team i is playing against team j on day t, then $X_{ijt}=1$. Otherwise, $X_{ijt}=0$.

Subject To:

<table>
<tr><th colspan="2">Teams Schedule Constraint</th></tr>
<tr><td colspan="2">Ask: We are asked by the Little League coach to abide by the constraint "Each team must play against every other team exactly twice."</td></tr>
<tr><th>Formulation</th><th>Implementation Into AMPL</th></tr>
<tr><td>$$\sum_t x_{ijt} + x_{jit} = 2 \quad \forall i, j \text{ where } i \neq j$$
Explanation: For team i and another team j in TEAMS, team i play against team j for exactly 2 times in DAYS.</td><td>subject to

Schedule_Constraint {i in TEAMS, j in TEAMS: i != j}:

sum {t in DAYS} (x[i, j,t] + x[j, i, t]) = 2;</td></tr>
<tr><td colspan="2">Explanation: This constraint ensures that each team competes with every other team exactly twice during the season, as required by the Little League coach. The mathematical formulation given on the left represents the sum of games played between teams i and j over all possible days t. Here, x_{ijt} is our binary decision variable that equals 1 if team i plays against team j on day t, and 0 otherwise. The constraint sums these variables for each unique pair of teams across all days. The sum must equal 2, indicating that exactly two matches are played between each pair of teams throughout the season.

In the AMPL implementation, Schedule_Constraint enforces this rule by iterating over all pairs of different teams (i, j) and all days t in the DAYS set. The expression sum {t in DAYS} (x[i, j, t] + x[j, i, t]) = 2; calculates the total number of games between team i and team j, ensuring it equals exactly two. This satisfies the requirement for a double round-robin tournament where each team plays every other team twice but also guarantees that the schedule is balanced and fair, with no team having an advantage or disadvantage in terms of the number of matches played against any other team.</td></tr>
</table>

<table>
<tr><th colspan="2">One Game Per Weekday Constraint</th></tr>
<tr><td colspan="2">Ask: We are asked by the Little League coach to abide by the constraint “There is at most one game per weekday.”</td></tr>
<tr><th>Formulation</th><th>Implementation Into AMPL</th></tr>
<tr><td>$\sum_{i,j} x_{ijt} \leq 1 \quad \forall t \text{ on weekdays}$

Explanation: For all t in DAYS that are not SATURDAYS, for team i and team j in TEAMS that i != j, team i can play against team j for at most once.</td><td>One_Game_Per_Weekday_Constraint {t in DAYS diff SATURDAYS}:

sum {i in TEAMS, j in TEAMS: i != j} x[i,j,t] <= 1;</td></tr>
<tr><td colspan="2">Explanation: This constraint specifies that on any given weekday, not including Saturdays, there should be no more than one game involving each pair of teams. This is implemented by summing all pairs of different teams i, j for each day within the DAYS set, excluding the set of SATURDAYS, ensuring that the sum of games for any match pairing/day does not exceed 1. Thus, it allows for only one game per day between any two teams to ensure there are no scheduling overlaps during the regular weekdays.</td></tr>
</table>

<table>
<tr><th colspan="2">Four-Day Period Constraint</th></tr>
<tr><td colspan="2">Ask: “No team should participate more than once within any four-day period of one week.”</td></tr>
<tr><th>Formulation</th><th>Implementation Into AMPL</th></tr>
<tr><td>i $\in TEAMS, d \in \{1, 6, 11, 16, 20, 25, 30, 35, 40, 45, 49\}\}$:
$\sum_{j \in TEAMS, t \in d..min(d+3,50) j \neq i} (x_{ijt} + x_{jit}) \leq 1;$

Explanation: For all team i in TEAMS and d in the set of start days, for team j such that j !=1, team i can play against team j for at most one time from day d to min(d+3, 50).</td><td>Four_Day_Period_Constraint {i in TEAMS, d in {1, 6, 11, 16, 20, 25, 30, 35, 40, 45, 49}}:

sum {j in TEAMS, t in d..min(d+3, 50): j != i} (x[i,j,t] + x[j,i,t]) <= 1;</td></tr>
</table>

Explanation: Since our group didn't directly import the excel spreadsheet, we arranged all the starts of four-day periods in one set; this constraint will make sure that from any of the start days to 3 days after that start day in 1 week, one team can play at most one game. For the last week of the schedule where there are less than 4 days, we'll use day 50, which is the last game day, as the end day of the Four-Day Period. The Four-Day period doesn't include Saturdays. In the AMPL model, Four_Day_Period_Constraint is applied to each team i and for specific starting days d within the season. We use the summation symbol for the games for team i against all other teams j within the range from day d to the third day after d (inclusive), or up to day 50, whichever comes first. We have expressed this by the range d..min(d+3, 50), where min(d+3, 50) ensures that the scheduling does not go beyond the last day of the competition, which is day 50. Our constraint also includes the condition j != i to prevent a team from being counted against itself.

The sum (x[i,j,t] + x[j,i,t]) in our code adds up the matches where team i plays against team j and other way around on the specified days. We set this sum less than or equal to 1 in order to mandate that team i plays at most one game within the four-day window. This will uphold the league's requirement that teams have at least a three-day rest period within a week.

Saturday Competition Constraint

Asked: "There's a weekly Saturday competition where four matches happen simultaneously"

Formulation	Implementation Into AMPL
$\sum_{i \in TEAMS, j \in TEAMS, t \in SATURDAYS: i \neq j} x_{ijt} = 4;$ Explanation: For all t in SATURDAYS, for team i, j in TEAMS such that i!=j, there are 4 games happening on each day t.	Saturday_Competition_Constraint1 {t **in** SATURDAYS}: **sum** {i **in** TEAMS, j **in** TEAMS: i != j} x[j,i,t] = 4;

Explanation: In the AMPL implementation, our constraint is applied to each t in the SATURDAYS set. Our sum function adds up the occurrences of games between each unique pair of teams (excluding games where a team would play against itself), ensuring that the total number of games on any given Saturday is exactly 4.

Non-Overlap Constraints

Non-Overlap Constraints are additions to the Four-Day Period Constraint. We are using the constraints below to make sure no team plays more than one game on one day.

<table>
<tr><th>Formulation</th><th>Implementation Into AMPL</th></tr>
<tr><td rowspan="3">$\{\mathrm{i} \in TEAMS, j \in TEAMS, k \in TEAMS, t \in DAYS : i \neq j \wedge j \neq k \wedge i \neq k\} :$
$x_{ikt} + x_{jkt} \leq 1$

$\{\mathrm{i} \in TEAMS, j \in TEAMS, k \in TEAMS, t \in DAYS : i \neq j \wedge j \neq k \wedge i \neq k\} :$
$x_{kij} + x_{kjt} \leq 1$

$\{\mathrm{i} \in TEAMS, j \in TEAMS, k \in TEAMS, t \in DAYS : i \neq j \wedge j \neq k \wedge i \neq k\}$
$x_{ijt} + x_{jit} \leq 1$

For all t in DAYS, team i, j, k in TEAMS where i != j, i!=k, j!=k:
1. Team k can play at most one game on day t as the second team that is playing in one game (sum will be ≤ 1).
2. Team k can play at most one game on day t as the first team that is playing one game (sum will be ≤ 1).
3. Team i can play against team j for at most 1 time on day t (sum will be ≤ 1).</td><td>Non-Overlap Constraint #1

Non_Overlap_Constraint1 {i in TEAMS, j in TEAMS, k in TEAMS, t in DAYS: i != j && j != k && i != k}:

x[i,k,t] + x[j,k,t] <= 1;</td></tr>
<tr><td>Non-Overlap Constraint #2

Non_Overlap_Constraint2 {i in TEAMS, j in TEAMS, k in TEAMS, t in DAYS: i != j && j != k && i != k}:

x[k,i,t] + x[k,j,t] <= 1;</td></tr>
<tr><td>Non-Overlap Constraint #3

Non_Overlap_Constraint3 {i in TEAMS, j in TEAMS, t in DAYS: i != j}:

x[i,j,t] + x[j,i,t] <= 1;</td></tr>
<tr><td colspan="2">Explanation: The Non-Overlap Constraints above help us to make sure each team plays only at most once in one day, and there is at most one game played by a combination of two teams in one day. By doing so, we avoid cases where a pair of teams play twice in one day and one team plays against two teams in one day.

Non-Overlap Constraint #1 is for potential overlaps where a team could be scheduled as the second team in two different games on the same day. Our code is to show that for any team k and any other team i (where i and j are the same), there can only be one game involving team k on day t, either against team i or any other team.

Non-Overlap Constraint #2 is a counterpart to the previous one, ensuring that no team is scheduled as the first team in more than one game on any given day. It is for situations where team k could be listed first against two different teams on the same day, which is not allowed.

Finally, Non-Overlap Constraint #3 deals with the direct matching between 2 teams, ensuring that team i does not play against team j more than once on day t. This is to prevent a scenario where the same 2 teams would play against each other twice in a day, which would violate the given rules.</td></tr>
</table>

<table>
<tr><th colspan="2">Total Output Constraint</th></tr>
<tr><td colspan="2">Ask: We are asked to create a schedule that aligns with “the requirement for a total of 72 competitions (36 pairs) to be held.”</td></tr>
<tr><th>Formulation</th><th>Implementation Into AMPL</th></tr>
<tr><td>$\sum_{i \in TEAMS, j \in TEAMS, t \in DAYS} x_{i,j,t} = 72;$

Explanation: The sum of all games x for team i, team j in TEAMS, t in DAYS is exactly 72.</td><td>Total_Output_Constraint:

sum {i in TEAMS, j in TEAMS, t in DAYS} x[i, j, t] = 72;</td></tr>
<tr><td colspan="2">Explanation: We wrote this to guarantee that the schedule is able to contain all games, so the schedule can only generate 72 games total. For this, we just set the sum of all decision variables (game played by team i vs team j on day t) equal to 72, since all decision variables will be binary.</td></tr>
</table>

Part 2

```
set TEAMS := 1..9;
set DAYS := 1..50;
set SATURDAYS := {5, 10, 15, 24, 29, 34, 39, 44};

var x{TEAMS, TEAMS, DAYS} binary;

subject to

Schedule_Constraint {i in TEAMS, j in TEAMS: i != j}:
  sum {t in DAYS} (x[i, j,t] + x[j, i, t]) = 2;

Total_Output_Constraint:
  sum {i in TEAMS, j in TEAMS, t in DAYS} x[i, j, t] = 72;

One_Game_Per_Weekday_Constraint {t in DAYS diff SATURDAYS}:
  sum {i in TEAMS, j in TEAMS: i != j} x[i,j,t] <= 1;

Four_Day_Period_Constraint {i in TEAMS, d in {1, 6, 11, 16, 20, 25, 30, 35, 40, 45, 49}}:
  sum {j in TEAMS, t in d..min(d+3, 50): j != i} (x[i,j,t] + x[j,i,t]) <= 1;

Saturday_Competition_Constraint1 {t in SATURDAYS}:
  sum {i in TEAMS, j in TEAMS: i != j} x[j,i,t] <= 4;

Non_Overlap_Constraint1 {i in TEAMS, j in TEAMS, k in TEAMS, t in DAYS: i != j && j != k && i != k}:
  x[i,k,t] + x[j,k,t] <= 1;

Non_Overlap_Constraint2 {i in TEAMS, j in TEAMS, k in TEAMS, t in DAYS: i != j && j != k && i != k}:
  x[k,i,t] + x[k,j,t] <= 1;

Non_Overlap_Constraint3 {i in TEAMS, j in TEAMS, t in DAYS: i != j}:
  x[i,j,t] + x[j,i,t] <= 1;
  solve;
```

Changed Constraints (Adjusted From Part 1)

<table>
<tr><th colspan="2">Saturday Competition Constraint</th></tr>
<tr><td colspan="2">Asked: Reduce the number of games on Saturday by changing the strict Saturday Competition Constraint.</td></tr>
<tr><td>Formulation</td><td>Implementation Into AMPL</td></tr>
<tr><td>$\sum_{i \in TEAMS, j \in TEAMS i \neq j} x_{j,i,t} \leq 4 \quad \forall t \in SATURDAYS;$

Explanation: For all t in SATURDAYS, the total number of games, for team i and a different team j in set TEAMS, is equal or less than 4 games.</td><td>Saturday_Competition_Constraint1 {t in SATURDAYS}:

sum {i in TEAMS, j in TEAMS: i != j} x[j,i,t] <= 4;</td></tr>
<tr><td colspan="2">Explanation: As required, we changed the Saturday Competition Constraint’s equation so the constraint changed from having exactly 4 games on each Saturday to having at most 4 games on each Saturday. Previously, the model required exactly four games to be played each Saturday. The revised formulation now allows for up to four games, introducing flexibility into the schedule.</td></tr>
</table>

Part 3

```
set TEAMS := 1..9;
set DAYS := 1..50;
set SATURDAYS := {5, 10, 15, 24, 29, 34, 39, 44};

var x{TEAMS, TEAMS, DAYS} binary;

subject to

Schedule_Constraint {i in TEAMS, j in TEAMS: i != j}:
  sum {t in DAYS} (x[i, j,t] + x[j, i, t]) = 2;

Total_Output_Constraint:
  sum {i in TEAMS, j in TEAMS, t in DAYS} x[i, j, t] = 72;

Exactly_One_Game_Per_Weekday_Constraint {t in DAYS diff SATURDAYS}:
  sum {i in TEAMS, j in TEAMS: i != j} x[i,j,t] = 1;

Four_Day_Period_Constraint {i in TEAMS, d in {1, 6, 11, 16, 20, 25, 30, 35, 40, 45, 49}}:
  sum {j in TEAMS, t in d..min(d+3, 50): j != i} (x[i,j,t] + x[j,i,t]) <= 1;

Saturday_Competition_Constraint1 {t in SATURDAYS}:
  sum {i in TEAMS, j in TEAMS: i != j} x[j,i,t] = 4;

Non_Overlap_Constraint1 {i in TEAMS, j in TEAMS, k in TEAMS, t in DAYS: i != j && j != k && i != k}:
  x[i,k,t] + x[j,k,t] <= 1;

Non_Overlap_Constraint2 {i in TEAMS, j in TEAMS, k in TEAMS, t in DAYS: i != j && j != k && i != k}:
  x[k,i,t] + x[k,j,t] <= 1;

Non_Overlap_Constraint3 {i in TEAMS, j in TEAMS, t in DAYS: i != j}:
  x[i,j,t] + x[j,i,t] <= 1;

solve;
```

Changed Constraints (Adjusted From Part 1)

<table>
<tr><th colspan="2">Exactly One Game Per Weekday Constraint</th></tr>
<tr><td colspan="2">Asked: Generate a schedule that there is exactly one game played on every possible weekday, and all the other constraints are same as the ones in part 1.</td></tr>
<tr><th>Formulation</th><th>Implementation Into AMPL</th></tr>
<tr><td>$\sum_{i \in TEAMS, j \in TEAMS i \neq j} x_{i,j,t} = 1 \quad \forall t \in DAYS \setminus SATURDAYS;$

Explanation: For all t in DAYS that are not in SATURDAYS, there is only one game happening on day t, where team i plays against a distinct team j.</td><td>Exactly_One_Game_Per_Weekday_Constraint {t in DAYS}:

sum {i in TEAMS, j in TEAMS: i != j} x[i,j,t] = 1;</td></tr>
<tr><td colspan="2">Explanation: For part 3, we were asked to generate a schedule with exactly one game every weekday, instead of at most 1 game every weekday. This means that we now have to ensure</td></tr>
</table>

that there is one game happening every weekday. So, we modify the inequality part of our "One Game Per Weekday Constraint" and change it to "Exactly One Game Per Day Constraint" by changing the (≤ 1) portion of the original constraint to (=1).

GENERAL RATIONALE FOR MODELING STRATEGY

To summarize, the modeling strategy we approached the scheduling request with is aimed at combining all decision variables and constraints into a simple but rational pattern that reflects scheduling specifications from the Little League's manager. Besides complying with the rules of the given letter & competition, our strategy reduces the complexity of the problem by simplifying it through integer programming techniques.

---

4. RESULTS & ANALYSIS

Part 1: AMPL Output	**Scheduling Interpretation of Output**
Objective = find a feasible point. ampl: display {j in 1.._nvars: _var[j] > 0} (_varname[j], _var[j]); : _varname[j] _var[j] := 170 'x[1,4,20]' 1 458 'x[2,1,8]' 1 464 'x[2,1,14]' 1 760 'x[2,7,10]' 1 830 'x[2,8,30]' 1 917 'x[3,1,17]' 1 937 'x[3,1,37]' 1 953 'x[3,2,3]' 1 979 'x[3,2,29]' 1 1105 'x[3,5,5]' 1 1315 'x[3,9,15]' 1 1365 'x[4,1,15]' 1 1416 'x[4,2,16]' 1 1442 'x[4,2,42]' 1 1457 'x[4,3,7]' 1 1494 'x[4,3,44]' 1 1755 'x[4,9,5]' 1 1774 'x[4,9,24]' 1 1804 'x[5,1,4]' 1 1839 'x[5,1,39]' 1 1884 'x[5,2,34]' 1 1895 'x[5,2,45]' 1 1923 'x[5,3,23]' 1 1962 'x[5,4,12]' 1	All variables on the left that = 1 would be matches that are scheduled within the season. As a refresher, We set binary variable X_{ijt} as the match between team i against team j on day t. If team i is playing against team j on day t, then $X_{ijt}=1$. With all conditions met from part 1, some of the following matches scheduled for the season would be: - Day 20: Team 1 vs Team 4 - Day 8: Team 2 vs Team 1 - Day 14: Team 2 vs Team 1 - Day 10: Team 2 vs Team 7 - Day 30: Team 2 vs Team 8 - Day 17: Team 3 vs Team 1 - Day 37: Team 3 vs Team 1 - Day 3: Team 3 vs Team 2 - Day 29: Team 3 vs Team 2 - Day 5: Team 3 vs Team 5 - Day 15: Team 3 vs Team 9 - Day 15: Team 4 vs Team 1 - Day 16: Team 4 vs Team 2 - Day 42: Team 4 vs Team 2

```
1994  'x[5,4,44]'  1
2206  'x[5,9,6]'  1
2260  'x[6,1,10]'  1
2291  'x[6,1,41]'  1
2322  'x[6,2,22]'  1
2324  'x[6,2,24]'  1
2365  'x[6,3,15]'  1
2377  'x[6,3,27]'  1
2432  'x[6,4,32]'  1
2447  'x[6,4,47]'  1
2469  'x[6,5,19]'  1
2499  'x[6,5,49]'  1
2601  'x[6,8,1]'  1
2661  'x[6,9,11]'  1
2746  'x[7,1,46]'  1
2750  'x[7,1,50]'  1
2755  'x[7,2,5]'  1
2839  'x[7,3,39]'  1
2843  'x[7,3,43]'  1
2860  'x[7,4,10]'  1
2879  'x[7,4,29]'  1
2926  'x[7,5,26]'  1
2931  'x[7,5,31]'  1
2959  'x[7,6,9]'  1
2985  'x[7,6,35]'  1
3102  'x[7,9,2]'  1
3134  'x[7,9,34]'  1
3178  'x[8,1,28]'  1
3179  'x[8,1,29]'  1
3215  'x[8,2,15]'  1
3274  'x[8,3,24]'  1
3298  'x[8,3,48]'  1
3334  'x[8,4,34]'  1
3338  'x[8,4,38]'  1
3390  'x[8,5,40]'  1
3394  'x[8,5,44]'  1
3439  'x[8,6,39]'  1
3463  'x[8,7,13]'  1
3471  'x[8,7,21]'  1
3633  'x[9,1,33]'  1
3634  'x[9,1,34]'  1
3686  'x[9,2,36]'  1
3694  'x[9,2,44]'  1
3705  'x[9,3,5]'  1
3810  'x[9,5,10]'  1
3874  'x[9,6,24]'  1
3979  'x[9,8,29]'  1
3989  'x[9,8,39]'  1;
```

- Day 7: Team 4 vs Team 3
- Day 44: Team 4 vs Team 3
- Day 5: Team 4 vs Team 9
- Day 24: Team 4 vs Team 9
- Day 4: Team 5 vs Team 1
- Day 39: Team 5 vs Team 1
- Day 34: Team 5 vs Team 2
- Day 45: Team 5 vs Team 2
- Day 23: Team 5 vs Team 3
- Day 12: Team 5 vs Team 4
- Day 44: Team 5 vs Team 4
- Day 6: Team 5 vs Team 9
- Day 10: Team 6 vs Team 1
- Day 41: Team 6 vs Team 1
- Day 22: Team 6 vs Team 2
- Day 24: Team 6 vs Team 2
- Day 15: Team 6 vs Team 3
- Day 27: Team 6 vs Team 3
- Day 32: Team 6 vs Team 4
- Day 47: Team 6 vs Team 4
- Day 19: Team 6 vs Team 5
- Day 49: Team 6 vs Team 5
- Day 1: Team 6 vs Team 8
- Day 11: Team 6 vs Team 9
- Day 46: Team 7 vs Team 1
- Day 50: Team 7 vs Team 1
- Day 5: Team 7 vs Team 2
- Day 39: Team 7 vs Team 3
- Day 43: Team 7 vs Team 3
- Day 10: Team 7 vs Team 4
- Day 29: Team 7 vs Team 4
- Day 26: Team 7 vs Team 5
- Day 31: Team 7 vs Team 5
- Day 9: Team 7 vs Team 6
- Day 35: Team 7 vs Team 6
- Day 2: Team 7 vs Team 9
- Day 34: Team 7 vs Team 9
- Day 28: Team 8 vs Team 1
- Day 29: Team 8 vs Team 1
- Day 15: Team 8 vs Team 2
- Day 24: Team 8 vs Team 3
- Day 48: Team 8 vs Team 3
- Day 34: Team 8 vs Team 4
- Day 38: Team 8 vs Team 4
- Day 40: Team 8 vs Team 5
- Day 44: Team 8 vs Team 5
- Day 39: Team 8 vs Team 6
- Day 13: Team 8 vs Team 7
- Day 21: Team 8 vs Team 7
- Day 33: Team 9 vs Team 1
- Day 34: Team 9 vs Team 1
- Day 36: Team 9 vs Team 2
- Day 44: Team 9 vs Team 2
- Day 5: Team 9 vs Team 3
- Day 10: Team 9 vs Team 5

	- Day 24: Team 9 vs Team 6 - Day 29: Team 9 vs Team 8 - Day 39: Team 9 vs Team 8

Part 2: AMPL Output	**Scheduling Interpretation of Output**
Objective = find a feasible point. ampl: display {j in 1.._nvars: _var[j] > 0} (_varname[j], _var[j]); : _varname[j] _var[j] := 87 'x[1,2,37]' 1 155 'x[1,4,5]' 1 179 'x[1,4,29]' 1 304 'x[1,7,4]' 1 360 'x[1,8,10]' 1 374 'x[1,8,24]' 1 439 'x[1,9,39]' 1 460 'x[2,1,10]' 1 567 'x[2,3,17]' 1 575 'x[2,3,25]' 1 697 'x[2,5,47]' 1 724 'x[2,6,24]' 1 781 'x[2,7,31]' 1 784 'x[2,7,34]' 1 812 'x[2,8,12]' 1 856 'x[2,9,6]' 1 929 'x[3,1,29]' 1 942 'x[3,1,42]' 1 1059 'x[3,4,9]' 1 1094 'x[3,4,44]' 1 1135 'x[3,5,35]' 1 1164 'x[3,6,14]' 1 1200 'x[3,6,50]' 1 1245 'x[3,7,45]' 1 1265 'x[3,8,15]' 1 1334 'x[3,9,34]' 1 1405 'x[4,2,5]' 1 1410 'x[4,2,10]' 1 1582 'x[4,5,32]' 1 1636 'x[4,6,36]' 1 1678 'x[4,7,28]' 1 1694 'x[4,7,44]' 1 1734 'x[4,8,34]' 1 1746 'x[4,8,46]' 1 1761 'x[4,9,11]' 1 1765 'x[4,9,15]' 1 1813 'x[5,1,13]' 1 1827 'x[5,1,27]' 1 1889 'x[5,2,39]' 1 1922 'x[5,3,22]' 1 1969 'x[5,4,19]' 1	All variables on the left that = 1 would be matches that are scheduled within the season. As a refresher, We set binary variable X_{ijt} as the match between team i against team j on day t. If team i is playing against team j on day t, then X_{ijt}=1. With all conditions met from part 2, some of the following matches scheduled for the season would be: - Day 37: Team 1 vs Team 2 - Day 5: Team 1 vs Team 4 - Day 29: Team 1 vs Team 4 - Day 4: Team 1 vs Team 7 - Day 10: Team 1 vs Team 8 - Day 24: Team 1 vs Team 8 - Day 39: Team 1 vs Team 9 - Day 10: Team 2 vs Team 1 - Day 17: Team 2 vs Team 3 - Day 25: Team 2 vs Team 3 - Day 47: Team 2 vs Team 5 - Day 24: Team 2 vs Team 6 - Day 31: Team 2 vs Team 7 - Day 34: Team 2 vs Team 7 - Day 12: Team 2 vs Team 8 - Day 6: Team 2 vs Team 9 - Day 29: Team 3 vs Team 1 - Day 42: Team 3 vs Team 1 - Day 9: Team 3 vs Team 4 - Day 44: Team 3 vs Team 4 - Day 35: Team 3 vs Team 5 - Day 14: Team 3 vs Team 6 - Day 50: Team 3 vs Team 6 - Day 45: Team 3 vs Team 7 - Day 15: Team 3 vs Team 8 - Day 34: Team 3 vs Team 9 - Day 5: Team 4 vs Team 2 - Day 10: Team 4 vs Team 2 - Day 32: Team 4 vs Team 5 - Day 36: Team 4 vs Team 6 - Day 28: Team 4 vs Team 7 - Day 44: Team 4 vs Team 7 - Day 34: Team 4 vs Team 8 - Day 46: Team 4 vs Team 8 - Day 11: Team 4 vs Team 9 - Day 15: Team 4 vs Team 9

```
2079   'x[5,6,29]'    1
2191   'x[5,8,41]'    1
2255   'x[6,1,5]'     1
2268   'x[6,1,18]'    1
2315   'x[6,2,15]'    1
2421   'x[6,4,21]'    1
2458   'x[6,5,8]'     1
2626   'x[6,8,26]'    1
2679   'x[6,9,29]'    1
2698   'x[6,9,48]'    1
2720   'x[7,1,20]'    1
2844   'x[7,3,44]'    1
2924   'x[7,5,24]'    1
2934   'x[7,5,34]'    1
2989   'x[7,6,39]'    1
2990   'x[7,6,40]'    1
3057   'x[7,8,7]'     1
3088   'x[7,8,38]'    1
3149   'x[7,9,49]'    1
3224   'x[8,2,24]'    1
3255   'x[8,3,5]'     1
3353   'x[8,5,3]'     1
3433   'x[8,6,33]'    1
3630   'x[9,1,30]'    1
3652   'x[9,2,2]'     1
3739   'x[9,3,39]'    1
3815   'x[9,5,15]'    1
3844   'x[9,5,44]'    1
3910   'x[9,7,10]'    1
3966   'x[9,8,16]'    1
3973   'x[9,8,23]'    1;
```

- Day 13: Team 5 vs Team 1
- Day 27: Team 5 vs Team 1
- Day 39: Team 5 vs Team 2
- Day 22: Team 5 vs Team 3
- Day 19: Team 5 vs Team 4
- Day 29: Team 5 vs Team 6
- Day 41: Team 5 vs Team 8
- Day 5: Team 6 vs Team 1
- Day 18: Team 6 vs Team 1
- Day 15: Team 6 vs Team 2
- Day 21: Team 6 vs Team 4
- Day 8: Team 6 vs Team 5
- Day 26: Team 6 vs Team 8
- Day 29: Team 6 vs Team 9
- Day 48: Team 6 vs Team 9
- Day 20: Team 7 vs Team 1
- Day 44: Team 7 vs Team 3
- Day 24: Team 7 vs Team 5
- Day 34: Team 7 vs Team 5
- Day 39: Team 7 vs Team 6
- Day 40: Team 7 vs Team 6
- Day 7: Team 7 vs Team 8
- Day 38: Team 7 vs Team 8
- Day 49: Team 7 vs Team 9
- Day 24: Team 8 vs Team 2
- Day 5: Team 8 vs Team 3
- Day 3: Team 8 vs Team 5
- Day 33: Team 8 vs Team 6
- Day 30: Team 9 vs Team 1
- Day 2: Team 9 vs Team 2
- Day 39: Team 9 vs Team 3
- Day 15: Team 9 vs Team 5
- Day 44: Team 9 vs Team 5
- Day 10: Team 9 vs Team 7
- Day 16: Team 9 vs Team 8
- Day 23: Team 9 vs Team 8

- *"Elaborate whether this would make a difference in the schedule you found in part 1."*

Looking through the schedules outputted from parts 1 and 2 of our AMPL code, there is no doubt that the adjustment of the Saturday games restriction from exactly 4 to at most 4 has affected the game scheduling. The change in the constraint is the reason behind varying play days for some matches and the different distribution of games in the season. To illustrate this point, in Part 1, Team 1 plays against Team 4 on Day 20 and in Part 2, this pairing is listed on day 5. The differences point to a more flexible scheduling process that allows for variations in the number of Saturday games and consequently impact the arrangement of games on other days.

	Actually, this flexibility may reduce the logistical challenges the league faces and increase the number of teams available due to their schedule.

Part 3: AMPL Output	**Scheduling Interpretation of Output**
CPLEX 22.1.1.0: integer infeasible. 244 MIP simplex iterations 0 branch-and-bound nodes No basis. Objective = find a feasible point. ampl: display {j in 1.._nvars: _var[j] > 0} (_varname[j], _var[j]); :_varname[j] _var[j] := ;	In Part 3 of our analysis using AMPL, we discovered that it's impossible to create a schedule where each weekday hosts exactly one game, while also meeting the constraints set in the earlier parts. Specifically, in Part 1, we scheduled four games for every Saturday, with games on all weekdays except for April 5th (day 18) and April 24th (day 37). However, to comply with Part 3's requirements, we need to add an additional game on both day 18 and day 37. This adjustment leads to a total of 74 games, exceeding our maximum limit of 72 games. This indicates an over-constrained problem, where our combination of constraints - like the total number of games, number of mandatory games on each day, and fixed holidays - makes it impossible to devise a feasible schedule that satisfies all these conditions simultaneously.

Ultimate Question: Is there a perfect solution to this problem, or will there always be an exception given the specified criteria and constraints?
Conclusion: Evaluating all our results from our AMPL model in hand given the scheduling specifications from the manager's letter, we can assert that while a feasible solution to the scheduling problem can be found, it may not *always* perfectly align with all the desired constraints; this means that the feasibility of the schedule is dependent on how restrictive the given constraints are, as illustrated by Parts 1,2, and 3. Parts 1 and 2 of our AMPL model output demonstrate that there exist feasible schedules, when working within the constraints of having exactly four games on Saturdays or at most four games, for example. However, Part 3's requirement to have a game on every possible weekday proved to be infeasible, showing us that the additional constraint in part 3 is way too restrictive. Therefore, the manager George Herman Ruth's initial question in his letter can be answered as follows: a perfect solution, defined as a solution that remains the same regardless of how the constraints are changed, is not possible for the Little League scheduling problem. If we simply focus on parts 1 and 2, we can see that we are able to generate feasible optimal schedules under certain constraints.

However, part 3, including the additional requirement of a game on every possible weekday, shows to us that a feasible schedule is not possible without relaxing or altering at least one of the constraints. This nuance suggests that when constraints are added or changed, the solution space is also affected, which means that there is no perfect, one-size-fits-all solution. For our given problem, the constraints dictate the feasibility of the solutions, and any change in these constraints requires us to reevaluate the solution, meaning that the perfect solution is conditional on the constraints imposed.

---

5. APPENDIX

Below is our AMPL code for parts 1-3.

Part 1	**Part 2**	**Part 3**

set TEAMS := 1..9;
set DAYS := 1..50;
set SATURDAYS := {5, 10, 15, 24, 29, 34, 39, 44};
 	
var x{TEAMS, TEAMS, DAYS} binary;
 	
subject to
 	
Schedule_Constraint {i in TEAMS, j in TEAMS: i != j}:
  sum {t in DAYS} (x[i, j,t] + x[j, i, t]) = 2;
 	
Total_Output_Constraint:
  sum {i in TEAMS, j in TEAMS, t in DAYS} x[i, j, t] = 72;
 	
One_Game_Per_Weekday_Constraint {t in DAYS diff SATURDAYS}:
  sum {i in TEAMS, j in TEAMS: i != j} x[i,j,t] <= 1;
 	
Four_Day_Period_Constraint {i in TEAMS, d in {1, 6, 11, 16, 20, 25, 30, 35, 40, 45, 49}}:
  sum {j in TEAMS, t in d..min(d+3, 50): j != i} (x[i,j,t] + x[j,i,t]) <= 1;
 	
Saturday_Competition_Constraint1 {t in SATURDAYS}:
  sum {i in TEAMS, j in TEAMS: i != j} x[j,i,t] = 4;
	set TEAMS := 1..9;
	set DAYS := 1..50;
	set SATURDAYS := {5, 10, 15, 24, 29, 34, 39, 44};
	
	var x{TEAMS, TEAMS, DAYS} binary;
	
	subject to
	
	Schedule_Constraint {i in TEAMS, j in TEAMS: i != j}:
	  sum {t in DAYS} (x[i, j,t] + x[j, i, t]) = 2;
	
	Total_Output_Constraint:
	  sum {i in TEAMS, j in TEAMS, t in DAYS} x[i, j, t] = 72;
	
	One_Game_Per_Weekday_Constraint {t in DAYS diff SATURDAYS}:
	  sum {i in TEAMS, j in TEAMS: i != j} x[i,j,t] <= 1;
	
	Four_Day_Period_Constraint {i in TEAMS, d in {1, 6, 11, 16, 20, 25, 30, 35, 40, 45, 49}}:
	  sum {j in TEAMS, t in d..min(d+3, 50): j != i} (x[i,j,t] + x[j,i,t]) <= 1;
	
	Saturday_Competition_Constraint1 {t in SATURDAYS}:
	  sum {i in TEAMS, j in TEAMS: i != j} x[j,i,t] <= 4;
		set TEAMS := 1..9;
		set DAYS := 1..50;
		set SATURDAYS := {5, 10, 15, 24, 29, 34, 39, 44};
		
		var x{TEAMS, TEAMS, DAYS} binary;
		
		subject to
		
		Schedule_Constraint {i in TEAMS, j in TEAMS: i != j}:
		  sum {t in DAYS} (x[i, j,t] + x[j, i, t]) = 2;
		
		Total_Output_Constraint:
		  sum {i in TEAMS, j in TEAMS, t in DAYS} x[i, j, t] = 72;
		
		Exactly_One_Game_Per_Weekday_Constraint {t in DAYS diff SATURDAYS}:
		  sum {i in TEAMS, j in TEAMS: i != j} x[i,j,t] = 1;
		
		Four_Day_Period_Constraint {i in TEAMS, d in {1, 6, 11, 16, 20, 25, 30, 35, 40, 45, 49}}:
		  sum {j in TEAMS, t in d..min(d+3, 50): j != i} (x[i,j,t] + x[j,i,t]) <= 1;
		
		Saturday_Competition_Constraint1 {t in SATURDAYS}:
		  sum {i in TEAMS, j in TEAMS: i

Non_Overlap_Constraint1 {i in TEAMS, j in TEAMS, k in TEAMS, t in DAYS: i != j && j != k && i != k}:
x[i,k,t] + x[j,k,t] <= 1;
 	
Non_Overlap_Constraint2 {i in TEAMS, j in TEAMS, k in TEAMS, t in DAYS: i != j && j != k && i != k}:
x[k,i,t] + x[k,j,t] <= 1;
 	
Non_Overlap_Constraint3 {i in TEAMS, j in TEAMS, t in DAYS: i != j}:
x[i,j,t] + x[j,i,t] <= 1;
solve;
	Non_Overlap_Constraint1 {i in TEAMS, j in TEAMS, k in TEAMS, t in DAYS: i != j && j != k && i != k}:
	x[i,k,t] + x[j,k,t] <= 1;
	
	Non_Overlap_Constraint2 {i in TEAMS, j in TEAMS, k in TEAMS, t in DAYS: i != j && j != k && i != k}:
	x[k,i,t] + x[k,j,t] <= 1;
	
	Non_Overlap_Constraint3 {i in TEAMS, j in TEAMS, t in DAYS: i != j}:
	x[i,j,t] + x[j,i,t] <= 1;
	solve;
		!= j} x[j,i,t] = 4;
		
		Non_Overlap_Constraint1 {i in TEAMS, j in TEAMS, k in TEAMS, t in DAYS: i != j && j != k && i != k}:
		x[i,k,t] + x[j,k,t] <= 1;
		
		Non_Overlap_Constraint2 {i in TEAMS, j in TEAMS, k in TEAMS, t in DAYS: i != j && j != k && i != k}:
		x[k,i,t] + x[k,j,t] <= 1;
		
		Non_Overlap_Constraint3 {i in TEAMS, j in TEAMS, t in DAYS: i != j}:
		x[i,j,t] + x[j,i,t] <= 1;
		
		solve;

Our direct raw output for our AMPL code for Parts 1-3 was given in the Results section.