SCHEDULING THE TURKISH SOCCER LEAGUE USING MATHEMATICAL PROGRAMMING

A Thesis Submitted to
the Graduate School of Engineering and Sciences of
İzmir Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of

MASTER OF SCIENCE

in Computer Engineering

by
Faruk GÜNEY

July 2013
İZMİR

We approve the thesis of **Faruk GÜNEY**

Examining Committee Members:

_____

Instr. Dr. Burak Galip ASLAN
Department of Computer Engineering, İzmir Institute of Technology

_____

Assoc. Prof. Dr. Aybars UĞUR
Department of Computer Engineering, Ege University

_____

Assist. Prof. Dr. Mustafa ÖZUYSAL
Department of Computer Engineering, İzmir Institute of Technology

31 July 2013

_____

Instr. Dr. Burak Galip ASLAN
Supervisor, Department of Computer Engineering
İzmir Institute of Technology

_____

Prof. Dr. Sıtkı AYTAÇ
Head of the Department of Computer Engineering

_____

Prof. Dr. R. Tuğrul SENGER
Dean of the Graduate School of Engineering and Sciences

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my thesis advisor, Instr. Dr. BURAK GALİP ASLAN, for all his encouragement and systematic guidance throughout the research, implementation and writing phases of this thesis.

I would also like to express great thanks to ÖZKAN GÜNDÜZ for all his support and motivation during this study.

I would especially like to thank to UĞUR MELEKE, who is a sport editor, for his inspiring articles in the newspaper that enlighten my study.

I would also extend my thanks to professionals in Veri Bilişim Hizmetleri San. ve Tic. A.Ş. for all their support during both of my work time and after my resignation.

I also would like to thank to TÜBİTAK (Türkiye Bilimsel ve Teknolojik Araştırma Kurumu) for their financial support (BIDEB-2210 Fellowship) during my MSc study.

The last but not the least, I would like to thank my family for their unconditional love and unwavering support.

ABSTRACT

SCHEDULING THE TURKISH SOCCER LEAGUE USING MATHEMATICAL PROGRAMMING

Generating a fair and feasible schedule is a difficult challenge for sports league organizers because of having various requirements from various involved parties. Some of these requirements are fairness requirements. Turkish Soccer League should be scheduled by according to these requirements especially because of the reduction of the confidence in Turkish Soccer League organizers and authorities due to the case of match fixing in recent years. As scheduling Turkish Soccer League, the prior requirements in our study are the minimization of the total number of break, carry over effect (COE) value of a schedule in addition to meeting the conflicting venue constraints. We decomposed scheduling process in phases to facilitate our solution. We used a different variation of *first-break-then-schedule* approach, proposed by Rasmussen and Trick (2008), to meet break conditions initially and solved each phase by applying different mathematical programming techniques including Integer Programming (IP) and Constraint Programming (CP). Our study generates a schedule having carry over effect (COE) value which is one of the lowest ones in European soccer competitions, in addition to minimizing total number of breaks.

ÖZET

TÜRKİYE FUTBOL LİGİ FİKSTÜRÜNÜN MATEMATİKSEL PROGRAMLAMA İLE OLUŞTURULMASI

Futbol ligi organizatörleri için adil ve makul bir fikstür hazırlamak, içerdiği gereksinimlerin çokluğu nedeniyle oldukça güçtür. Bu gereksinimlerden biri de tarafsızlık ilkesidir. Türkiye Futbol Lig'inde son yıllardaki şike davası süreci nedeniyle güven ortamı sarsılmıştır ve bu güven ortamının yeniden tesisi için her konuda tarafsızlık ile ilgili unsurların ön plana çıkartılmasına her zaman olduğundan daha fazla ihtiyaç duyulmaktadır. Bu çalışmamızda Türkiye Futbol Ligi'nin fikstürünü hazırlarken, fisktürün şu anki halinin bazı temel özelliklerini koruyarak, bir fikstürün tarafsızlık ölçütlerinden olan devreden etki değerini (*Carry Over Effect - COE*) ve toplam kırılım (*break*) değerini en aza indirmeyi amaçladık. Çözümümüzü hayata geçirmek için fikstür oluşturma sürecini bir kaç safhaya ayırdık. Öncelikle kırılım şartlarını karşılamak ve her safhayı, Tamsayı Programlama ve Kısıtlı Programlama gibi teknikleri içeren bir matematiksel programlama tekniği ile çözebilmek için 2008 yılında Rasmussen ve Trick'in de önermiş olduğu *first-break-then-schedule* (önce kırılımı hesapla – sonra fikstür oluştur) tekniğinin farklı bir varyasyonunu kullandık. Çalışmamız sonucunda minimum kırılım sayısına sahip bir fikstürü oluşturmamıza ek olarak, Avrupa'daki liglerin içerisinde düşük seviyede COE değerine sahip olan fikstürlerden birini elde ettik.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

Soccer is one of the most important sports without doubt and there is a huge market for different types of stakeholders in soccer games such as teams, broadcasting companies, fans, police etc. Turkish Soccer has an important role in the European Soccer. According to Deloitte Annual Review of Football Finance (Sports Business Group, 2012) Turkish Super League is the 7th biggest revenue generating league with the €515 million annually (RR). A major broadcasting company in Turkey (Digiturk) have been paying $321 million for the broadcast rights since 2010 and Turkish Soccer League has more broadcasting value than many other countries (%44 of total revenue) (RR). Before 2010, this value was $140 million. This significant increase in investment illustrates the rising interest in Turkish Soccer, so the scheduling of games in Turkish Soccer League becomes more important. Schedule is in a tight bond with game attendance, public interest, and profitability of events for sponsors, broadcasters and advertisers. Furthermore, schedules have an obvious impact on the results of the competition itself. Hence involved parties want to organize a schedule that maximizes their revenue by taking into account the circumstance that the organization is attractive, fair, practicable and safe for anyone involved. For meeting these requirements of competitions, organizers of soccer leagues should be wary about scheduling because good schedule is a crucial for an effective competition.

Organization of a feasible schedule is not an easy task because of many constraints originating from various stakeholders. Some of these constraints may be conflicting. Although some of these constraints are common for majority of competitions, there may be additional requirements giving priority to special constraints desired by national federations. For example in Italy, multiple television companies have broadcasting rights and the schedule emphasized on the fair distribution of important matches over the rounds for each TV station (Della Croce and Oliveri, 2006). Although researchers from mathematics, computer science and operational research studied on this subject in last decade, there are quite a few papers that propose solution approaches on specific soccer leagues. Although there are many rumors about match fixing especially in Turkey where soccer industry possesses an important economic

power, there has not been any academic studies published on **League Scheduling.**

This thesis is organized as follows; first, the general review of scheduling concept including its terminology has been introduced with its basic problems (constraints) and applications in other countries. Next, we present the current Turkish Super League scheduling system and algorithmic alternatives for a solution. After that, we present our solution strategy for Turkish Super League by focusing on mathematical programming. Then, we conclude by analyzing impact of our solution method and prospects for future work.

CHAPTER 2

SPORT SCHEDULING TERMINOLOGY

A league is a sports competition in which **n teams** play against each other according to a given timetable. Usually, n is an even number but sometimes it may be an uneven number. The league organizes **games** between teams. Games are scheduled in **rounds**. Each round is played on a given **day**. A schedule consists of games assigned to rounds. A schedule is called as **compact schedule**, if each team plays one game in each round. A schedule is **relaxed schedule**, if each team plays more than one game in any round. If a team has no game in a round, it is called as **bye** in that round (Nurmi et al., 2010).

Teams have an associated **venue (stadium)**. If a team plays match in its associated stadium, it is called as a **home match** for that team; otherwise it is called as an **away match** for that team. The sequence of home matches, away matches or byes for a team is defined as the **home-away pattern (pattern)** for that team. If a team plays two or more consecutive home matches or away matches rounds, team's pattern is said to have a **break** in that rounds. Two patterns are called as complementary patterns, if the first pattern has an away game and the second pattern has a home game or if the first pattern has a home game and the second pattern has an away game for all rounds.

1	**0**	**1**	**0**	**1**
0	1	0	1	0

(a)

Rounds	1	2	3	4	5
Team 1	1	0	1	0	1
Team 2	0	1	0	1	0
Team 3	1	0	1	1	0
Team 4	0	1	0	0	1
Team 5	1	0	0	1	0
Team 6	0	1	1	0	1

(b)

Table 2.1. (a) Two complementary patterns (b) Sample pattern set for a 6 teamed league

Table 2.1 (a) shows two complementary patterns for a compact single round robin tournament and 2.1 (b) shows a sample pattern set for 6 the league consisting of 6 teams. In these tables home matches are illustrated as **1** and away matches are illustrated as **0.**

Round Robin is an algorithm for scheduling where all participant teams play each other team in turn. If all teams play each other once, it is called as **Single Round Robin Schedule (1RR)**. If they play twice, it is named as **Double Round Robin Schedule (2RR)**. Each couple of teams play one match at home stadium and one match at away stadium. Else if they play each other four times, it is said as **Quadruple Round Robin Schedule (4RR)**. Each couple of teams plays 2 matches home and 2 matches away. Table 2.2 shows the sample 2RR tournament information about number of rounds and matches.

# of Rounds :	2*(n-1) **Ex:** For Turkish Super League = 2*(18 -1) =34
# of Games :	(n /2)*(n-1)*2 **Ex:** For Turkish Super League => (18/2) * (18-1)*2 = 306

Table 2.2. Double Round Robin Tournament Calculations for n Teams

There is a term which is called as **Mirrored Double Round Robin (M2RR)**. In M2RR, teams play against all of their opponents in first n-1 rounds. After single round robin finished, they play them in a same row at reverse venues. Table 2.3 shows the sample M2RR schedule.

There is a term which is called as **Canonical Schedule**. In canonical schedule (de Werra, 1980), teams play against same opponents in a same row. Table 2.4 shows sample fixture part of Turkish Super League 2012/2013 season. Sivasspor, Fenerbahçe and Trabzonspor have same opponents in a row (Eskişehirspor-Gençlerbirliği-Kayserispor) which is presented in bold letters in Table 2.4.

R1	R2	R3	R4	R5	R6	R7	R8	R9	R10
1-5	1-4	1-2	1-6	1-3	5-1	4-1	2-1	6-1	3-1
2-3	2-5	3-4	2-4	2-6	3-2	5-2	4-3	4-2	6-2
4-6	3-6	5-6	3-5	4-5	6-4	6-3	6-5	5-3	5-4

Table 2.3. Sample Mirrored Double Round Robin Schedule (M2RR)

	R28	**R29**	**R30**	**R31**	**R32**
Sivasspor	**Eskişehirspor**	**Gençlerbirliği**	**Kayserispor**	İstanbul Bld.	Galatasaray
Fenerbahçe	Orduspor	**Eskişehirspor**	**Gençlerbirliği**	**Kayserispor**	İstanbul Bld.
Trabzonspor	Akhisar Bld.	Orduspor	**Eskişehirspor**	**Gençlerbirliği**	**Kayserispor**

Table 2.4. Sample of Real Canonical Schedule

CHAPTER 3

SPORT SCHEDULING PROBLEMS (CONSTRAINTS)

Sport Scheduling Problem consists of many constraints. Researchers who study on this subject aims to find an optimal schedule which meets particular constraints depending on requirements of associations which varies from one country to another. Global constraints are mandatory for every competition in which schedule is Round Robin. Typical constraints are constraints which can be different for different competitions according to their special requirements. We summarize the global and typical constraints of this problem in this chapter. Later, we focus on some of these constraints which are break and carry-over effect value minimization. We can briefly list the global constraints of round-robin schedule as below.

Global Constraints (Guillermo Dur´an et al., 2007)

- Each team play each of the others once over the course of the (n-1) rounds in the tournament for n teams, if n is not odd.
- Each team play one match per round.
- Each (n/2) games should be played, if n is not odd.

Typical constraints consist of large number of constraints which are arising from teams, TV networks, sports associations, fans and local communities. Some of these constraints naturally conflict. Thus, associations classify some of them as hard constraints and the rest of them as soft constraints depending on their priorities. We can group of typical constraints in subtitles which are listed as below.

Pattern Constraints (Guillermo Dur´an et al., 2007)

- Each team plays at most one sequence of n consecutive rounds at home or n consecutive round at away. This condition implies that no team plays more than n consecutive rounds at home or n consecutive rounds at away.
- Let A be a set of rounds which cannot have break. If a team plays at home (away) in

any round of A, it must play away (at home) in the following round. For example if A={1,16} for a league of 18 teams, none of patterns of any team can have any breaks in first or last rounds of the Round Robin.

Venue Constraints

- Balanced proportion of home and away games in a season for all teams. The difference between the number of played home and away games for each team must not be larger than k in any stage of the tournament.

Example: If k = 2 and round 20, any of team should be played at most 11 home matches or at most 11 away matches.

- A team should play home or away in a certain round. This kind of constraint is normally imposed when a venue is unavailable due to other events.

Example: Because of 19th May celebrations at Atatürk 19 Mayıs Stadium, teams of Ankara should play away in this round.

Complementary Constraints (Rasmussen, 2006)

- If two teams share same venue, these teams should have complementary patterns. When one of these teams plays at home, the other team should play at away in same rounds.

Example: Two teams should have patterns like Table 2.1 (a)

- Dependency between the leagues. Lower-division teams may not wish to play at home at the same time as a neighboring higher-division team to attract more supporters and get higher revenue.

Example: Gaziantep Büyükşehir Belediyespor, which play in PTT 1st League in Turkey may not wish to play at home, when Gaziantepspor, which play in Turkish Super League, plays at home in any round.

Time and Distance Constraints (Anson and Lester, 2007)

- Match days and their start time should be fair for all teams depending on their European schedule.

Example: In 2012/2013 season Fenerbahçe plays in EURO Cup, thus it plays European matches on Thursdays and Galatasaray plays in Champions League, thus it plays European matches on Tuesdays or Wednesdays. For this reason, after European matches Galatasaray plays national league matches on Fridays or Saturdays and Fenerbahçe plays league matches on Sundays or Mondays. To provide fairness in these clubs challenge, association should balance match day differences.

- Some teams may wish to play home games after European matches to reduce tiredness.

Game Constraints (Rasmussen, 2006)

- These are constraints which are fixing or prohibiting games in a certain round because of reason such as TV broadcaster requirements, security etc.

Example 1: Fenerbahçe-Galatasaray is a rival game and Digiturk (Broadcast Company of Turkey) this game to be scheduled in last rounds of the season. They expect more advertisement revenue to increase competency in championship.

Example 2: Police requires rival matches not to be scheduled in May the 1st for security.

Strength Group of Teams Constraints

- Teams should not play more than ***k*** consecutive matches against teams in same strength groups.
- There should be at most ***m*** games between the teams in strength group **s** between rounds ***r1*** and ***r2*** for balanced spread of games against top teams over the season.

Geographical Constraints

- Teams wish to play consecutive away games against opponents in near cities to reduce expenses and minimize tiredness of team.
- Teams do not wish to play consecutive away games which need long trip for them to prevent tiredness of team.

In many applications the constraints are classified into hard constraints and soft constraints. All hard constraints must be satisfied in a feasible solution, while the soft constraints are penalized such that penalties are incurred if the constraints are violated. In addition to minimizing the number of violated soft constraints, the objective of a sports scheduling problem in our study is to minimize either the number of breaks and carry over effect value of a schedule. In the following two sections, we will give more detailed information about minimization of breaks and carry-over effect value than other constraints.

3.1. Break Minimization Constraint

If a team plays two consecutive home matches or two consecutive away matches, break emerges. It is one of the most important constraints of sport scheduling problem. League organizers demand schedules with minimum number of breaks of all

teams as total or balanced number of breaks for all teams. (i.e., schedules in which all teams have the same number of breaks) (Ribeiro, 2012). This problem can be presented in mathematical model like below.

For each team and for each pair of consecutive round, a 0-1 variable B_{ij} is defined. B_{ij} = 1 means that the **team i** has a break involving the games play at round j and at round j + 1. These variables are called break variables. For each team there are n-2 break variables (Regin, 2001).

Our objective function is:

$$\textbf{Min} \sum_{i=1..n}\sum_{j=1..n-2} B_{ij} \qquad (3.1)$$

This function provides minimization of total number of breaks of all teams in schedule (Regin, 2001).

Sports scheduling problems are usually solved by following one of two decomposition approaches.

- **First - schedule, then break:** Determine each round's games in advance. After that define home-away patterns of teams according to pre-defined schedule.
- **First – break, then schedule:** Firstly define all teams' home-away patterns, next assign games to rounds according to these patterns.

For different scheduling problem sets, these approaches have been studied by researchers but break minimization problems are usually taken into consideration by researchers who follow one of both of approaches. First study on this subject was by de Werra (1980; 1981; 1982; 1988). Later, this context has been discussed by Brouwer et al. (2008), Miyashiro et al. (2003), Miyashiro and Matsui (2003) and Post and Woeginger (2006). Optimization and constraint programming approaches for break minimization have been presented by Regin (2001) and Rasmussen and Trick (2007).

3.2. Carry-Over Effect Value Minimization Constraint

Carry-Over Effect generally refers to the possible effect on the performance of a team at some stage of a sports tournament due to a specific event that occurred during a previous stage in the tournament (MP Kidd, 2010). We say that a **team i** gives a carry-over effect to a **team j**, if some other **team t**'s game against **team i** is followed by

a game against **team j**. For example, if **team i** is a very strong team, we can easily say that **team t** will be in worse shape to play next round against **team j**. Because teams, which play against strong opponents, will be more tired or may be faced with injuries because of making big effort. Thus, **team j** will be advantageous. Moreover, the carry-over effect could also be in a strong relationship with psychological form of team. When **team t** loses confidence and morale after a severe loss against the strong **team i**, again to the benefit of their next opponent, **team j**. The opposite may be true if **team i** is a weak team. The main purpose of the studies on this subject is balancing carry-over effect between teams.

The **carry-over effects value minimization problem** aims to find a schedule of which carry-over effect **(COE)** value is minimum and this value is one of most important indices of quality of round-robin schedule. Definition of COE will be presented below.

It is said that team i gives a carry-over effect to team j if a team plays i in round s then j in round s + 1 (s ∈ {1, 2, . . . , n − 1}; regard round n as round 1) (Miyashiro and Matsui, 2006). Rounds are considered cyclically. The last round (n-1) is followed by the first round and first round may be considered as round n.

For a given schedule, the **carry-over effects matrix** *C* (coe-matrix for short) is a non-negative matrix whose element c_{ij} denotes the number of carry-over effects given by ***team i*** to ***team j*** in the schedule. By its definition, every coe-matrix satisfies the following (Miyashiro and Matsui, 2006):

• The sum of each row is $n - 1$;

• The sum of each column is $n - 1$;

• Every diagonal element is 0.

T/W	1	2	3	4	5
1	2	3	5	4	6
2	1	5	4	6	3
3	4	1	6	5	2
4	3	6	2	1	5
5	6	2	1	3	4
6	5	4	3	2	1

(a)

j/i	1	2	**3**	4	5	6
1	0	3	**1**	1	0	0
2	0	0	**1**	0	1	3
3	1	1	**0**	1	1	1
4	0	1	**1**	0	3	0
5	3	0	**1**	0	0	1
6	1	0	**1**	3	0	0

(b)

Table 3.1. (a) Sample Schedule with 6 Teams (b) COE Matrix (6x6) of Schedule

Our objective function is :

$$\textbf{min} \sum_{i,j} (c_{ij})^2 \quad \text{(Russell, 1980)} \qquad (3.2)$$

COE value of the schedule in table 3.1 (a) is calculated as below.

$$\sum_{i,j} (c_{ij})^2 = 5*(3)^2 + 15*(1)^2 + 16*(0)^2 = 60$$

If all columns of COE matrix would be like 3rd column which is written in bold, minimum COE value of round robin tournaments with 6 teams can be calculated as n*(n-1) which can is shown below.

$$\sum_{i,j} (c_{ij})^2 = 30*(1)^2 + 6*(0)^2 = 30 \text{ equals to}$$
$$n*(n-1) = 6*5=30$$

A schedule which achieves lowest carry-over effect value is called a "**balanced schedule".** Russell (1980) proposed a construction algorithm which achieves balanced schedule, when the number of teams is power of 2. For other values of n (except n=12), Anderson (1999) method achieved the best known results. For 12 teams, best known results are by Guedes and Ribeiro (2009), who use heuristic for solving carry over effects minimization problem. Trick (2001) developed a constraint programming method to prove Russell's method's optimality for n=6. Miyashiro and Matsui (2006) developed a time-consuming heuristic based on random permutations of the rounds of fixtures created by the polygon method (Kirkman, 1847). They reported huge computation times as well.

CHAPTER 4

SOCCER SCHEDULE APPLICATIONS

In this chapter, we give an outline of European Soccer Leagues' scheduling applications in real world in first section. Then we introduce current Turkish Super League's scheduling method and problems related to this method in second section. In the end, we give brief information about academic studies that present solutions for specific soccer leagues.

4.1. European Soccer Schedules

In this section, we give the general information about schedules of top 10 European Soccer Leagues according to UEFA country coefficients of 2012/2013 season ("Country Coefficients 2012/13", 2013). This ranking assessment is derived from participating national clubs' success in European Champions League and European League for the last 5 years. When we look at these competitions for 2012/2013 season, the number of teams of schedules varies between 16 (Ukraine, Russia and Portugal) and 20 (Spain, England, Italy and France). There is not an extra rule for larger or more populated countries have more teams in their leagues. For example, Russian League has fewer teams than Turkish League. No competition is played with an odd number of teams. However, we can easily detect that top ranking leagues according to the UEFA country coefficients have more teams than other leagues.

These 10 leagues are organized in a double round robin schedule. In some competitions play-off stage follows the regular stage of the competition. No play-off schedule can be decided in the start of the season, because teams of play-off are determined in accordance with their regular stage ranking of the league. Play-offs can determine which team will be champion, which team will qualify for European Tournaments (Euro Cup or Champions League), which team will be relegated or which team will promote. If there is at least one team in play-off stage from lower league it is termed as **Promotion Play-Off Stage**, although there are teams from Premier Division to avoid relegation. If all teams in play-off stage are from Premier Division and play to

avoid relegation, we defined it as **Relegation Play-Off Stage**. In the Table 4.1., it can be seen that 2 (Germany, Netherlands) of 10 leagues have play-off stage after regular one. In Germany, promotion play-off stage is played to promote Bundesliga. In Netherlands, the teams ranked between 6 and 9 take part in a play-off to deserve final Euro Cup ticket as well as teams ranked 16 and 17, which take part in promotion play-off with the teams from lower division to avoid relegation. Thus, number of relegation teams changes in Germany and Netherlands according to results of play-off stage.

Country	**N**	**Format**	**Play- Off Stage**				**Rounds**	**# of Rel.**
			Title	**Europe**	**Rel.**	**Prom.**		
Spain	20	2RR	No	No	No	No	38	3
England	20	2RR	No	No	No	No	38	3
Germany	18	2RR	No	No	No	Yes	34	2-3
Italy	20	2RR	No	No	No	No	34	3
France	20	2RR	No	No	No	No	34	3
Portugal	16	2RR	No	No	No	No	30	2
Ukraine	16	2RR	No	No	No	No	30	2
Russia	16	2RR	No	No	No	No	30	2
Netherlands	18	2RR	No	Yes	No	Yes	34-40	1-3
Turkey	18	2RR	No	No	No	No	34	3

Table 4.1. UEFA Top Ranking 10 Leagues

When we focus on the round-robin stages of these leagues, we can see many differences in their applications. Most of those leagues are divided into 2 single round-robin tournaments. However, we can see in the 3rd column of Table 4.3 that English Premier League doesn't have 2 equal parts. It means that any team in the league, which has **n** teams, doesn't play with every other team once in first (**n-1**) rounds of the season. 1st column in same table shows that most of the top leagues in Europe are not organized in canonical way which is mentioned in Chapter 2. In the recent past, most of the competitions' organization was canonical but in last decade this situation has been changing, after mathematical methods have started to play an important role in scheduling. 4th column of the Table 4.1 illustrates symmetrical way of leagues and we

can see that there are different symmetrical schemes applied in these top leagues which are shown in the Table 4.2 (Goossens and Spieksma, 2012). Table 4.3 show that 2 of these leagues (England and Netherlands) don't apply any symmetrical scheme. Because of not having equal parts, it is impossible for England to apply any symmetrical scheme. Only Dutch League is not symmetrical, although having 2 equal parts.

Mirroring	1	2	3	...	n-1	1	2	3	...	n-2	n-1
French	1	2	3	...	n-1	2	3	4	...	n-1	1
English	1	2	3	...	n-1	n-1	1	2	...	n-3	n-2
Inverted	1	2	3	...	n-1	n-1	n-2	n-3	...	2	1

Table 4.2. Symmetry Schemes
(Source: Goossens and Spieksma, 2012)

Last column in Table 4.3 presents how many rounds league has at least between same opponents. These differences vary from one league to another league related to its symmetrical scheme. If any of these leagues would apply English or Inverted scheme, this value would be 1. However they don't apply one of these 2 schemes. Thus, minimal round difference emerges in asymmetrical leagues (England, Netherlands).

Country	**Canonical**	**Equal Parts**	**Symmetry**	**Round Diff.**
Spain	Yes	Yes	Mirror	19
England	No	No	None	6
Germany	No	Yes	Mirror	17
Italy	No	Yes	Mirror	19
France	No	Yes	French	18
Portugal	No	Yes	Mirror	15
Ukraine	Yes	Yes	Mirror	15
Russia	Yes	Yes	French	14
Netherlands	No	Yes	None	9
Turkey	Yes	Yes	Mirror	17

Table 4.3. UEFA Top Ranking 10 Leagues

Note: Round differences of England and Netherlands are taken by Goossens and Spieksma (2012)

Table 4.4. shows the values of our focus points of sports scheduling problem which are "break minimization" and "COE value minimization". There is only one common point in these leagues that teams should play maximum 2 home or 2 away consecutive matches. Other attributes of these leagues vary in the Table 4.4. 3rd column is the number of breaks which teams of league have. Leagues (Spain, Ukraine, Russia and Turkey), which apply canonical schedules achieve break minimization. It is easier to schedule all teams which haven't break at first round and at last round in fixtures which are organized in canonical way and have mirror scheme. The 4th column and the 5th column show which leagues have teams having break at first and at the end. Most of the leagues prefer not to have breaks beginning and finishing rounds of season. Although leagues with canonical schedules achieve break minimization, it fails in carry-over effect value minimization and maximizes this value. Because of highness of this value, fairness of tournament cannot be achieved. Therefore, 4 of 5 top league schedules are not organized in canonical way.

Country	Max. Series	Per Team	Begin	End	COE Value
Spain	2	0-3	No	No	5548
England	2	5-8	No	No	888
Germany	2	0-3	No	No	1100
Italy	2	0-4	No	No	884
France	2	2	No	Yes	1278
Portugal	2	0-3	No	No	650
Ukraine	2	0-3	Yes	No	2580
Russia	2	2	Yes	Yes	2580
Netherlands	2	4-9	No	No	668
Turkey	2	0-3	No	No	3876

Table 4.4. UEFA Top Ranking 10 Leagues

Note: Most of the data above are taken from the article by Goossens and Spieksma (2012) including countries Spain, England, Germany, Italy, France, Russia, Netherlands and Turkey.

4.2. Scheduling in Turkish League

In this section, we focus on Turkish Soccer League scheduling system and we will try to give its specifications in more detail. In previous section, general attributes of Turkish Soccer League have been depicted in tables while presenting European top league's attributes.

Turkish League has 18 teams. The scheduling of these teams is organized in 2RR format without play-off stage. Only in 2011-2012 season, play-off stage is tried to increase enthusiasm which was minimized during the case of match-fixing attempts. However, Turkish Football Federation gave up this application after this trial season.

2012 / 2013 season league organization was done as it was before 2011 / 2012 season. There is a pre-defined match schedule with numbers (1...18). This is called as **Basic Match Schedule (BMS)**. This schedule determines the matches (home-away assignments and opponents) in each round. This schedule is a canonical schedule which meets minimization of break and mirroring requirements. Table 4.5 shows the first 6 rounds of the season.

1	**2**	**3**	**4**	**5**	**6**
1-2	8-5	13-18	8-1	5-14	18-7
3-4	2-17	17-15	12-5	11-15	12-1
5-6	18-15	9-14	14-7	9-18	8-4
7-8	16-13	11-16	15-13	1-10	16-5
9-10	4-1	5-10	6-2	17-13	15-9
11-12	12-9	3-8	16-9	2-8	14-3
13-14	6-3	2-4	4-17	3-12	6-17
15-16	7-10	7-12	10-3	7-16	13-11
17-18	14-11	1-6	18-11	4-6	10-2

Table 4.5. BMS of Turkish League

Before the season kicks off, 1st round of the season is determined by assigning teams to number according to draws. This draw is not purely random. For example in 2010/2011 season, top teams (Fenerbahçe, Galatasaray and Beşiktaş) can only assign to

numbers (1,10 and 18) to meet some constraints such as preventing matches between these teams in first 4 rounds of the season (Uğur Meleke, 25th July 2010).

In 2012/2013 season, Turkish Football Federation gave up preventing to schedule matches between top teams in first 4 rounds. However, teams which play their home matches in the European part of Istanbul (Beşiktaş, Galatasaray, Kasımpaşa and İstanbul Büyükşehir Belediyespor) can only assign to number set (7,8,9 and 10) which include complementary patterns to meet security and traffic constraints.

In BMS, teams assigned to numbers (excluding number 17) follow same sequence of opponents. This sequence can be seen easily, if we take one team (number 13) as a reference. When we look at 1st round matches of the season and follow the numbers clockwise, we can find team 13's next rounds' opponents sequentially. Team 13's opponent set is sequentially {14, 16, 18, 15, 17, 11, 9, 7, 5, 3, 1, 2, 4, 6, 8, 10, 12, 14}. Number 17 has a **free fixture** which means it has not same sequence as other teams. Thus, while following the clockwise opponents' of number 13, we pass the number 17 after 18. We put number 17 as opponent sequence, when the clockwise number list comes to referenced number. Uğur Meleke illustrated this sequence by showing Gençlerbirliği's 2012/2013 season fixture in his article in Milliyet newspaper (Fikstür Çekimi Adil Miydi?, 19th July 2012).

4.3. Real-Life Scheduling Applications by Researchers

In this section, we give the general information about the academic studies, which are applied in specific leagues. In spite of large numbers of papers on league scheduling, there are only a few papers on specific leagues in Europe and South America.

Bartsch et. al (2006) presented heuristics and branch and bound to solve the scheduling problems of German Football Federation and Austrian Football Federation. German Football Federation only once used this approach but Austrian Football Federation applied this paper's solution for their league many times in practice. Della Croce and Oliveri (2006) applied integer programming to schedule Italian league, which has requirements such as conditions on home-away matches and additional cable tv requirements. However, their contact with Italian Football Federation is not tight for application in real world. Kendall (2008) presented a study on minimization of the

travel distances by English clubs over Christmas and New Year Period. Goossens and Spieksma (2009) presented mixed integer programming model to schedule Belgian First Division 2006/2007 season and next season they applied Two-Phased Approach, which is a variant of *'first break and then schedule'* approach (Rasmussen and Trick, 2008). The schedules were used in practice for mentioned seasons. Rasmussen (2008) presented an Integer Programming model which uses logic-based Benders decomposition and column generation to solve a triple round robin tournament for the Danish football league.

Duran et al. (2007) proposed an Integer Programming method as well to meet specific constraints of Chilean Soccer League, such as television station based on geography or strength group of teams. Since 2005, Chilean Soccer Association used this method with some improvements. Ribeiro and Urrutia (2007; 2009; 2010) presented an Integer Programming solution to schedule Brazilian league to meet break minimization and TV broadcasting revenue maximization constraints. Ribeiro and Urrutia (2011) reported that approach has been used in practice for 2009, 2010 and 2011 seasons of the tournament. Lastly in Honduras, an Integer Programming solution was presented by Fiallos et al. (2010) to schedule Honduras League which is played by 10 teams.

CHAPTER 5

MATHEMATICAL PROGRAMMING TECHNIQUES

Mathematical programming is an optimization model based on selecting the best alternative from a set of available options with the utilization of computer programs. It relies on probability theory and mathematical models make predictions about future events. In this chapter, we give an outline of mathematical techniques (Integer Programming and Constraint Programming) used in our study for scheduling the Turkish Soccer League.

5.1. Integer Programming

Integer Programming is a mathematical optimization and feasibility approach which consists of integer variables. It is a kind of Linear Programming method thus it refers to the Integer Linear Programming (ILP) in many studies.

Integer Programming (IP) is an efficient instrument to optimize and solve sports scheduling problems. Most of the Round Robin Tournaments problems are solved using applications of Integer Programming because formulation of Single Round Robin Tournament Problem is easier.

If n and r sequentially denote the number of teams and number of rounds in a Round Robin Tournament, following variable definitions are used by the majority of models:

$$x_{ijt} = \begin{cases} 1, & \text{If team } i \text{ plays at home against team } j \text{ in round } t, \\ 0, & \text{otherwise} \end{cases}$$

For teams i,j = 1, ..., n ($i \neq j$) and rounds t = 1,...,r. The Constraints for Double Round Robin tournament can be formulated as below. (Kendall et al., 2010)

$$\sum_{t=1}^{R} x_{ijt} = 1, \quad \forall 1 \leq i, j \leq n, i \neq j \ , \tag{5.1}$$

$$\sum_{j=1}^{N} x_{ijt} \leq 1, \quad \forall 1 \leq i \leq n, 1 \leq t \leq r \qquad (5.2)$$

The first constraint guarantees that every team must play every other team at home. The second ensures that every team plays once in each round.

There is variety of Integer Programming methods applied to sports scheduling problems. These methods are used to solve real league scheduling applications likewise to solve some theoretical problems of scheduling such as break minimization or traveling distance minimization. In the next part, we will give brief information about these methods for better understanding what they are.

5.1.1. Branch and Bound Algorithm

Branch and Bound is a general algorithm, which is a useful tool for many kinds of optimization problems. Although this technique is not utilized only for integer programming, it is counted as the backbone of the integer programming. Majority of the effective solutions for integer programming are developed based on this framework. This method's basic principle is *divide and conquer* which means firstly partitioning total set of feasible solutions into smaller solution subsets and then evaluating these smaller subsets until best solution is found.

Typically we can view this algorithm as a tree search. Initially, the tree has a single node, called as **root node**. Other nodes must be derived by branching of the root node during search. Each of the nodes has an associated Linear Programming (LP) problem. After solution of each associated LP problem, one of the four following possible situations emerge (Rasmussen, 2006).

i. The problem is infeasible
ii. The problem is feasible with an integer solution
iii. The problem is feasible with fractional solution and a solution value is worse than the current best integer solution
iv. The problem is feasible with fractional solution and a solution value is better than the current best integer solution

- If the first or third situation emerges, we can omit the node without further evaluation.

- If the second situation emerges, new solution value and the current best solution value are compared. If the new solution value is better than the current best one, the new one is stored as the current best one; else this node can be omitted as first and third situations.
- If the fourth situation emerges, suitable integer variable to fractional value is selected and branching continues on the value of this variable.

The algorithm progresses from node to node. When all nodes are visited, we either find the optimal solution or show that the problem is infeasible. For clarifying the understanding of this method, an example is given below.

Example: Consider the following problem ("Solving Integer Programming", n.d.).

Maximize $Z = 9x_1 + 5x_2 + 6x_3 + 4x_4$

Such that

$6x_1 + 3x_2 + 5x_3 + 2x_4 \leq 10$,

$x_3 + x_4 \leq 1$, $-x_1 + x_3 \leq 0$,

$-x_2 + x_4 \leq 0$, $x_i \leq 1$ for $1 \leq i \leq 4$ and $x_i \geq 0$

The general application of this algorithm is depicted in Figure 5.1 step by step to show detailed calculations of every node's Linear Programming problem itself. Finally we can see that the best value for Z is 14 with $x_1 = 1$, $x_2 = 1$, $x_3 = 0$ and $x_4 = 0$.

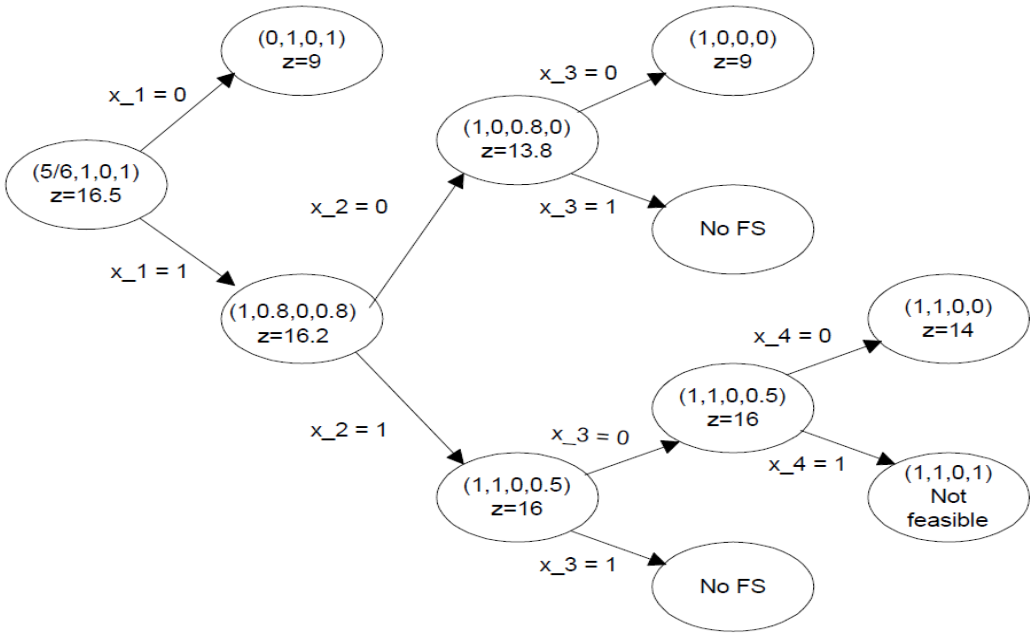


Figure 5.1. Sample Branch and Bound Solution
(Source: Solving Integer Programming, n.d.)

5.1.2. Branch and Cut Algorithm

Branch and Cut is an exact algorithm, which is a combination of **branch and bound algorithm** and **cutting plane method**. This algorithm facilitates the solution of the associated LP problems (see 5.1.1) of branch and bound algorithm in nodes by cutting off the infeasible solution area for IP problem.

Cutting Plane is a mathematical technique to reduce the bounds of solution space. These bounds are held by adding cuts to the problem. **A cut** is a valid inequality that eliminates some of the LP feasible region. **A valid inequality** is a constraint that doesn't eliminate any feasible solution for IP problems. Cutting Plane approach is not adequate to solve general IP problem. Thus, this method requires branching, which results in a branch and cut algorithm. The speed of branch and bound algorithm increases significantly with the addition of cutting plane approach. Because, cutting planes provide considerable reduction in the size of the search tree. In order to make things clear for this method, an example is given below.

Example: Consider the following problem.

Maximize $Z = 3x + 4y$ ("Cutting Plane Techniques", n.d.)

Such that

$5x + 8y \leq 24,\ x,y \geq 0$ and integer

Solution:

Step 1 :

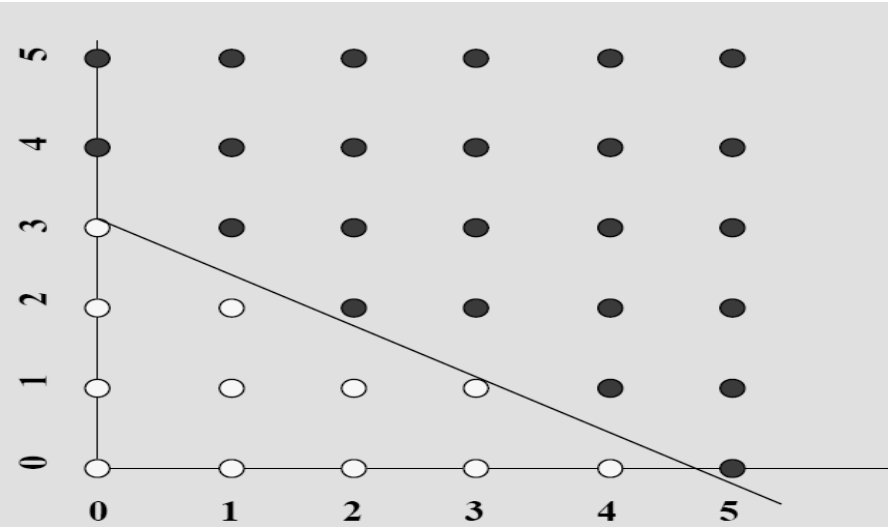


Figure 5.2. A Graph of LP Solution of the Problem
(Source: Cutting Plane Techniques, n.d.)

Figure 5.2 shows the feasible solution region for the problem in terms of given constraints. Consequently, the optimal LP solution set is for x, y, z is (4. 8, 0, 14.4).

Step 2:

We add a valid inequality to improve the quality of bounds by making the LP feasible region closer to IP feasible region.

New constraint: $x \leq 4$

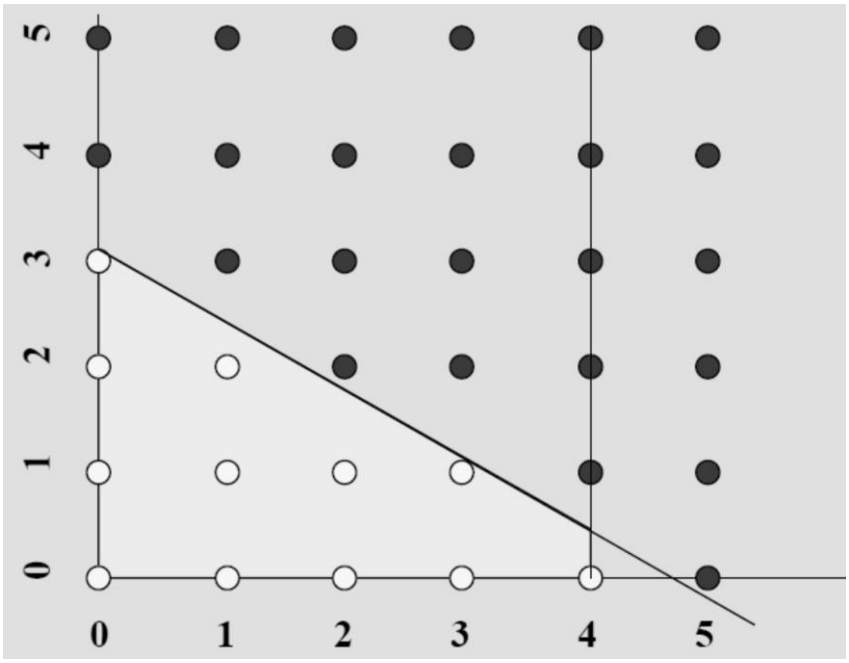


Figure 5.3. A Graph of LP Solution of the Problem
(Source: Cutting Plane Techniques, n.d.)

Figure 5.3 shows the feasible solution region for the problem in terms of given constraints and added constraint. The optimal LP solution set is for x, y, z is (4, 0.5, 14) sequentially.

Step 3:

We add one more valid inequality to improve the quality of bounds by making the LP feasible region closer IP feasible region.

New constraint: $x + y \leq 4$

Figure 5.4 shows the feasible solution region for the problem in terms of given constraints and newly added constraint. The optimal LP solution set is for x, y, z is (8/3, 4 /3, 13 1/3) sequentially. This bound is enough to establish that x = 3 and y = 1 is optimal for the IP. As a result, the solution set for x, y, z is **(3, 1, 13)**.

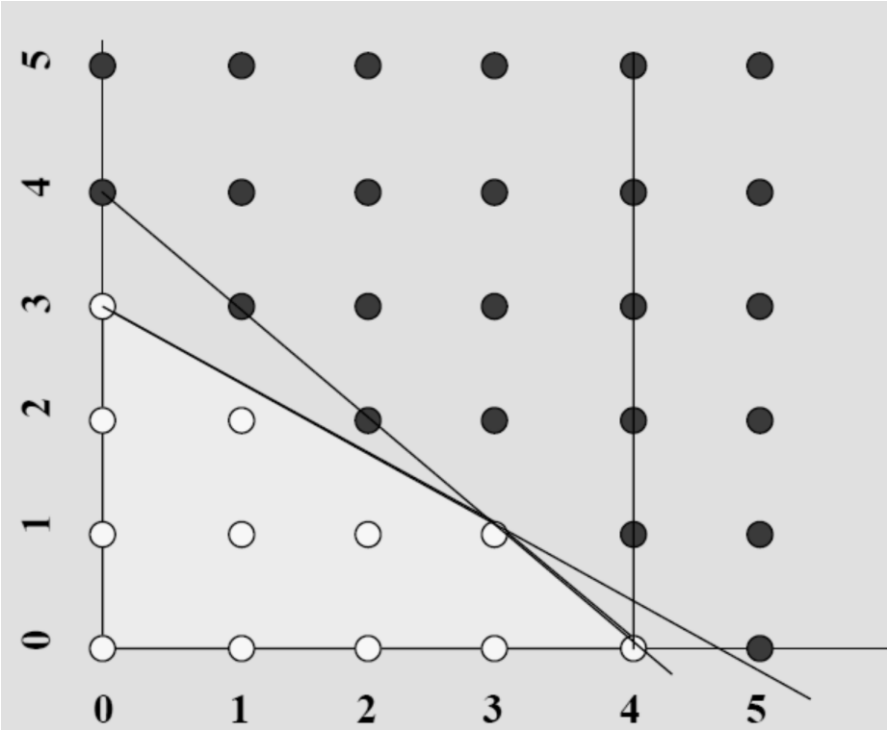


Figure 5.4. A Graph of LP Solution of the Problem
(Source: Cutting Plane Techniques, n.d.)

5.1.3. Benders Decomposition Approach

Benders Decomposition is an optimization model proposed by Benders (1962). This model works by decomposing the problem into two problems, namely the **master problem** and the **sub problem**. This decomposition is done by distinguishing primary variables from secondary variables. Primary variables consist of integer variables where secondary variables are the rest of the variables. Initially, the algorithm searches over the primary variables to solve the master problem. Next, it searches for a solution of sub problem over secondary variables for each trial value of master problem. If a possible solution is infeasible or suboptimal, the algorithm finds out why and adds a new constraint to eliminate possible violating variable values for the same reason. Added constraint is defined as **Benders Cut**. This cut enables learning from mistakes by generating a new row to the LP equation. After addition of Benders Cut, the master problem is resolved and this procedure is iterated until an optimum value is found (Rasmussen, 2006).

5.1.4. Branch and Price Algorithm

Branch and Price is an optimization algorithm which is an integration of a **branch and bound algorithm** and **column generation** approach to solve large-scale integer programming problems. In this algorithm, additional columns are added to Linear Programming problems by adding new variables to the equations, in contrast to **Benders** and **Branch and Cut** approaches.

This algorithm also partitions the problem into sub problems, which are defined as **Restricted Master Problem** and **Pricing Problem**. At the root node of the branching tree, restricted master problem is solved and most of the columns are removed. Next, pricing problem is used to improve existing columns because restricted master problem solution may not be optimal. If there are improving columns, these columns are added to the restricted master problem which will be optimized again in the next iteration. This procedure iterates until no profitable columns are found in pricing problem. When all of optimized columns are found for a node, the algorithm proceeds to the new node as described in branch and bound algorithm and problem decomposition is iterated for each node (Rasmussen, 2006).

5.2. Constraint Programming

Constraint Programming (CP) is an efficient solution technique to solve hard combinatorial optimization problems such as planning and scheduling. This technique emerged as a result of the occurrence of constraints in other research areas including Artificial Intelligence, Programming Languages, Symbolic Computing and Computational Logic. Constraints have been used systematically since 1980s. In constraint programming, the working process of the computational system bases on constraints described in chapter 3 and the idea of this method is satisfying those constraints.

Searching for a solution to meet the constraints is defined as **Constraint Satisfaction Problem (CSP)**. It is the problem which is modeled by using

- A finite set of variables,
- A finite domain for each variable,
- A finite set of constraints.

Each constraint adds restriction to the solution set for variables to take. A solution of CSP provides assignment to each variable without violating any of the constraints. It can be based on finding any solution or an optimum solution in terms of some criteria or exploiting all possible solutions by searching. This method is similar to IP approach but all of the constraints have to be solved using LP methodology in IP approach. CP problems can be formulated by intuitive models. We will present basic components of CSP and some of the algorithms working in this manner in next section.

5.2.1. Systematic Search Algorithm

Systematic Search means trying every possible candidate for a solution. There are many search algorithms to solve CSPs. Although these algorithms are not always efficient because of time complexity, they introduce basic algorithms which provide infrastructure for advanced algorithms.

Generate and Test (GT) Algorithm

Generate and Test Algorithm is the most straightforward constraint satisfaction algorithm which ensures to find a solution if there is any. It is an exhaustive search of the problem space by exploring all possible combination of variable assignments. The number of combinations is the size of the Cartesian product of all the variable domains. The workflow of this algorithm is given below.

1) Generation of a possible solution set by assigning values for all variables
2) Testing of constraints to check for violations.

If testing is successful, it means that a solution is found; otherwise another solution set is generated. This approach is time consuming, because there can be many wrong assignments of values which can be detected in the testing phase. Late inconsistency detection significantly slows down the performance. Thus, this algorithm is theoretically useful in simple problem domains (Bartak, 1995).

Backtracking (BT) Algorithm

Backtracking is a kind of systematic search algorithm which merges the generation and test phases of GT algorithm. It is the tree-search algorithm like Branch and Bound algorithm where variables are instantiated sequentially. Despite of this similarity, for each node, values are assigned to variables consistently with the other

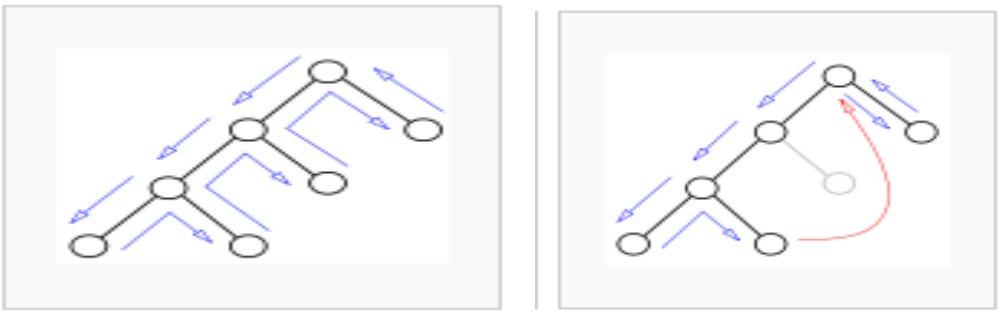
variables assigned before and validity of constraints is checked in this method. The size of the partial solution extends after every node processed until any inconsistency occurs or any feasible solution is obtained. If an assignment for any node violates any of the constraints, backtracking is done to the previous node. Backtracking to previous node iterates until a node that has available alternatives is found. When a node is found, assignments of other variables are triggered again originating from that node. Whenever a violation occurs in partial assignment, elimination of subspace from the Cartesian product of all variable domains is provided by backtracking. This property of backtracking provides crucial time saving to find a feasible solution compared to GT algorithm.

Although this algorithm is more efficient than GT algorithm, it suffers several disadvantages which are discussed as follows:

1. **Thrashing:** Thrashing means having no identification for conflicting values. This situation causes repeating failures due to same reasons. Clearly, it is instantiating variables to values which are infeasible for same reason. This failure can be avoided by **backjumping** method where backtracking is done directly to the variable that caused the failure. (Bartak, 1995)
2. **Redundant Work:** Redundant Work means having no memory for conflicting instantiation for variables. Therefore, same conflicting values can be assigned to same variables in different branches of the tree.
3. **Late Conflict Detection:** Inconsistency can be detected only after assigning values to variables. There is not intuition for conflicting values. This drawback can be avoided by applying **forward checking** consistency techniques to check the possible conflicts. (Haralick and Elliott, 1980).

Backjumping (BJ) Algorithm

Backjumping is a technique which eliminates infeasible search space. It is similar to BT algorithm but it allows going back more than one node. When BJ algorithm detects violation, it analyzes that which variables' assignments are conflicting for this violation. After the detection of those variables, BJ algorithm backtracks to the most recent conflicting variable (Gaschnig, 1979). Figure 5.5 shows the process difference of BT and BJ algorithms.



Backtracking Algorithm **Backjumping Algorithm**

Figure 5.5. Graphs of Backtracking and Backjumping Algorithms
(Source: Wikipedia)

Backmarking (BM) Algorithm

Backmarking is also a variant of BT technique which is beneficial to reduce the number of checks. It remembers conflicting instantiation for variables. Thus, it avoids rechecking of same constraints with same conflicting variables (Haralick and Elliott, 1980).

5.2.2. Consistency

If all assignments of variables can compose a part of a feasible solution for all constraints, these set of constraints are named as **consistent**. Thus, none of assignment for any variable violates any constraint and the variables can be instantiated without backtracking. However, obtaining such a degree is very hard; however search space for a problem solution can be reduced using some techniques which are defined as **Consistency Techniques**.

Consistency Techniques provide sooner detection of inconsistency between constraints. The following example illustrates the basic idea of the consistency techniques.

Example: Consider the following problem.

Find suitable solution sets for integer variables (x, y)

Such that

$x < y, \quad 6 \leq x \leq 10, \quad 2 \leq y \leq 8$

Solution: X's domain ranges from 6 to 10 and y's domain ranges from 2 to 8. We can prune search space by contracting domains of x and y which cannot be a part of consistent solutions for a constraint x < y. New domains are below.

x = {6, 7}, y = {7, 8} => Many inconsistent values are removed. However, all of remaining combinations of variables are not consistent. (For example x = 7 and y = 7 is not consistent.)

In next part, we present some of well-known consistency degrees which can be obtained using consistency techniques by removing the inconsistent values from the variables' domains in the constraint network. Before presenting consistency degrees, let us give the definition of *constraint graph*. **Constraint graph** is used to show the links between constraints in a constraint satisfaction problem (CSP).

Constraint graph for binary CSP problem have:

- Nodes representing variables
- Links representing the constraints

Node Consistency

It is the simplest form of consistency level. A variable is **node consistent** if all values within its domain are consistent with the all unary constraints on the variable. A CSP is **node consistent** if and only if all variables are node consistent.

Example: Consider the following variables if they are node consistent. D_X and D_Y are domains of variables (x, y) and C_X and C_Y are unary constraints on these variables.

$D_X = \{1, 2, 3, 4, 5\}$, $D_Y = \{-1, 1, 2\}$

$C_X = x < 6$, C_Y= y is a positive integer variable

Answer: x is node consistent because of all of values in its domain meets C_X constraint, however y is not node consistent because -1 is not a positive integer value and violates C_Y constraint.

Arc Consistency

If all values which are inconsistent with binary constraints are removed, these variables are **arc-consistent**. Clearly, the arc (V_i, V_j) is **arc-consistent** if and only if for each value x from the domain D_i there exists a value y in the domain Dj such that the assignment V_i =x and V_j = y satisfies all the binary constraints on V_i, V_j. A CSP is arc-consistent if and only if every arc in its constraint graph is arc-consistent (Bartak, 1995).

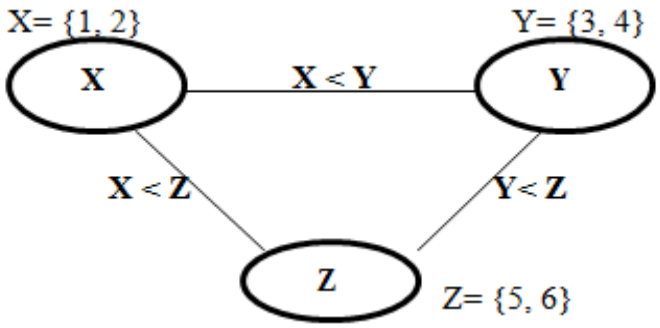


Figure 5.6. Sample Arc-consistent Graph

Figure 5.6 depicts the arc-consistent CSP. Although, arc-consistency enables removing many inconsistencies from constraint graph, there may be no feasible solution for domains of variables which are already arc-consistent. Figure 5.7 shows the arc-consistent CSP which does not have any solution meeting all constraints.

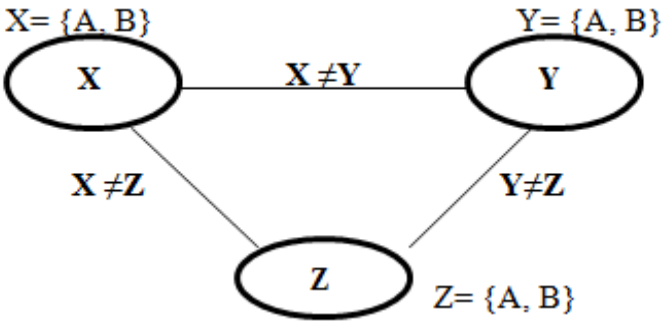


Figure 5.7. Sample Arc-consistent Graph Having No Solution.

K-Consistency

Figure 5.7 shows that although arc-consistency provides strong consistency, it is not sometimes enough to solve CSPs immediately without search. Thus, stronger consistency degree is needed to solve some problems.

A CSP is **K-Consistent** if and only if given any consistent assignment of *K-1* variables; there exists an assignment of any K^{th} variable such that the *K* values taken together satisfy all of the constraints among the *K* variables. A CSP is **strongly K-consistent** if it is J-consistent for all J<=K. Because of obtaining such a strong consistency degree generally requires huge exponential time, backtracking cannot be avoided to solve CSPs with great number of variables.

5.2.3. Constraint Propagation

In the previous parts of Constraint Programming section, systematic search algorithms and consistency techniques were introduced. Both of these techniques can be used to solve CSPs individually but combinations of them are preferred because of their own drawbacks in terms of efficiency. The technique using combinations of systematic search and consistency are defined as **Constraint Propagation**. In this technique, consistency checks are embedded in the search algorithm.

At each node in search algorithm, domains of variables are reduced using consistency techniques. These consistency techniques can be applied after instantiating a variable to provide consistency for already instantiated variables or to prevent possible conflicts of variables which are not instantiated yet. In the following section, general working mechanisms of these algorithms will be presented.

Backtracking (BT)

Backtracking algorithm was introduced in detail in section 5.2.1, thus we only give this algorithm's contribution to the consistency check in this section briefly. BT provides consistency among already instantiated variables by checking validity of constraints considering the partial instantiation. Backmarking and Backjumping introduced in 5.2.1 are intelligent backtracking algorithms.

Forward Checking (FC)

Forward Checking is an algorithm which detects possible future conflicts between variables before an inconsistent situation emerges. It is provided by re-arranging future variables' domains using consistency check between current variable and future variables, after a variable is instantiated. If any of the future variables' domains becomes empty after instantiating, we can easily deduce that current branch of the search tree will lead to inconsistency and this branch can be pruned so earlier than emergency of any inconsistency in backtracking. Although this operation requires more work than backtracking after each assignment of variable, it is expected to reduce total amount of search time by discarding important amount of branches before failures (Haralick and Elliott, 1980).

Look Ahead (LA)

In forward checking, restricted arc consistency is obtained because of checking consistencies between current instantiated variable and a future variable only. However, **Look Ahead** algorithm detects possible conflicts between future variables. Look Ahead prunes more amounts of search tree branches and reduces attempts but it can require much more time than forward checking to provide consistency between future variables. Thus, forward checking and backtracking are widely used to solve CSPs in applications (Haralick and Elliott, 1980).

5.2.4. Variable and Value Orders

So far, we have presented some algorithms regarding determination of variables and their domains. In addition to these, efficiency of tree search algorithm is also directly related to the ordering of variables and values of their domains. If a right value is chosen for each node, the solution can be found without any backtracking. In this section, we introduce the importance of orderings (Bartak, 1995).

Variable Ordering

Variable ordering has an important impact on the complexity of backtracking search. It can be actualized in 2-different ways:

1. **Static Ordering:** Order of the variables is determined before the search and it is not changed during the search.
2. **Dynamic Ordering:** Order of variables change during the search depending on the recent state of search.

Dynamic ordering is not useful for all search algorithms, because ordering after every instantiation of variable can be time consuming. Thus, dynamic ordering can be used in algorithms in which branches are reduced like the search algorithm using forward check. The basic notion in variable ordering is **Fail-First** principle which selects the variable first whose instantiation will lead to a failure. Fail-First principle gives these points priorities for ordering the variables.

- Prefers variables with smaller domain.
- Prefers most constrained variables.
- Prefers variables with more constraints than previous variables.

Value Ordering

When a variable is selected to be assigned a value from its domain, it is an important question that which particular value of domain will most likely lead to a solution. For decreasing the solution time, value ordering is used and basic notion of this ordering is **Succeed-First** principle which likely leads to a solution. Succeed-First principle gives these points priorities to order variables.

- Prefers values resulting in less domain reduction.
- Prefers values that can simplify the problem.

CHAPTER 6

SCHEDULING APPROACH FOR THE TURKISH SOCCER LEAGUE

Sport scheduling problems are generally solved using decomposition approaches in other leagues. This approach is based on dividing main problem into sub-problems which are solved sequentially. Some studies provide solutions using **first schedule and then break approach**, however some of them use **first break and then schedule approach**. First schedule and then break approach follows the procedure in which games are initially assigned to rounds before home-away pattern regulations of teams, in contrast to first break and then schedule approach. (Kendall et. al, 2010)

In Turkey, soccer league is scheduled using predefined match schedule which is organized in canonical 2RR league form. Although current schedule system provides break minimization and complementary pattern sets for teams sharing same stadiums, it fails in carry-over effect issue by having maximum COE value with 18 teams. The idea of this study is decreasing COE value of schedule in addition to minimizing breaks, and providing complementary pattern sets for teams sharing stadiums. Therefore, our schedule firstly should guarantee to meet the hard constraints which are already met in current schedule system. As a secondary goal, it should decrease COE value. We preferred to study on scheduling Turkish Soccer League using decomposition approach like the majority of other leagues, because we have constraints in which some of them are critical and some of them have less importance.

Our solution procedure is based on a variant of *first break and then schedule* approach (Rasmussen and Trick 2008), because constraints related to patterns is more important in this approach. We also organized our solution procedure based on 1RR tournament for even number of teams because current 2RR schedule in Turkey is mirrored and we can easily deduce the 2^{nd} round robin schedule by reversing home-away teams of the games. We divide solution procedure in 5 steps which are listed below,

1. Generate patterns having minimum breaks,
2. Find a feasible pattern set consisting of generated patterns,
3. Find a suitable timetable for selected pattern set,

4. Calculate COE value of the timetable and compare it with the current best value,
5. Assign teams to patterns.

Our solution procedure consists of these steps and each of them is solved using different mathematical methods including integer programming and constraint programming. We now describe each step with an emphasis on how it is used for the Turkish Soccer League in our method.

6.1. Pattern Generation with Minimum Breaks

Break minimization is one of the crucial constraints which must not be violated in our method. 1st column of the Table 4.4 shows that breaks have maximum 2 consecutive home or away matches. Obviously, any team cannot play more than 2 consecutive home or away matches. When we look at 1st column the same table, current Turkish League Basic Match Schedule has patterns which have maximum 3 breaks for a 34 round 2RR tournament. Thus, each pattern for 1RR should have maximum 1 break to prevent exceeding 3 breaks. Although we give chance to a user for determining maximum number of breaks for 1RR by using a variable for this purpose in our program, maximum number of consecutive home or away matches for break occurrence is assigned to 2 by default.

To define the patterns for n teams (where n is an even number) and rounds t = 1,...,n-1, the constraints for Single Round Robin tournament can be formulated by integer programming as below.

Problem 1: Generating patterns consisting of binary variables (x1, x2.., x_{n-1}) having less number of breaks than 2.

Solution: Let y_t be the binary variable for the break occurrence in round t and k be the variable for total number of breaks for a pattern and let z be the maximum number of breaks for a pattern.

Such that

$\forall\ x_t \in \{0, 1\}\ ,\ y \in \{0, 1\},\ z=1,$

$$x_t = \begin{cases} 1, & \text{if pattern has home match at round t ,} \\ 0, & \text{if pattern has away match at round t} \end{cases}$$

$$y_t = \begin{cases} 1, & \text{if pattern has consecutive home or away matches in rounds t and t+1} \\ 0, & \text{Else} \end{cases}$$

$$0 < \sum_{t=1}^{n-1} x_t + x_{t+1} + x_{t+2} < 3, \quad \forall\, t+2 \leq n-1 \qquad (6.1)$$

$$\sum_{t=1}^{n-2} y_t < z+1 \qquad (6.2)$$

The first set of constraints ensures that a pattern has maximum 2 consecutive home or away matches and the second set ensures that a pattern has maximum 1 break, when z =1. We solved this IP problem using Branch and Bound approach.

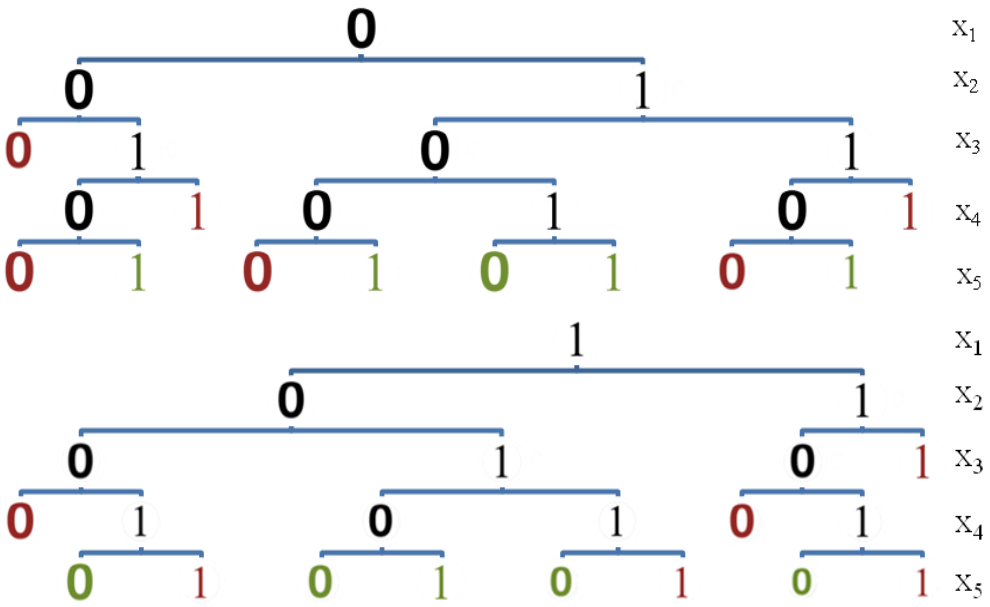


Figure 6.1.Pattern Generation Search Tree for 6-Team-League Using Branch and Bound

Figure 6.1 shows two Branch and Bound search trees for league with 6 teams. First tree is the tree whose first round is away game, in contrast to the second one. We use depth-first search to generate all of the suitable patterns meeting constraints. As a nature of Branch and Bound technique, nodes are derived by branching of the root node as search proceeds. x_1 is the root node for both trees. After branching, each node has a problem modeled with IP and consisting of the first and the second constraints. If node is feasible for these constraints, search proceeds until all nodes are visited and checked for constraints. In figure 6.1, numbers in red letter show the infeasible nodes to solve IP and numbers in green letter show the nodes which are the last nodes of the solution sets (desired patterns). We enumerate these patterns, after generation of each desired one.

Table 6.1 shows all of the suitable patterns for 6 teams with at most one break below. The maximum count of suitable patterns for n team with at most one break can be calculated with the formulas in the following.

# of Suitable Patterns without break	= 2
# of Suitable Patterns With 1- Break	= 2 * (n-2)
# of Suitable Patterns With 1-Break At Most	+ = 2 * (n-1)

Maximum n-2 different patterns can be generated with 1-break occurrence for away matches. The same situation is valid for 1-break occurrence for home matches as well. Intrinsically maximum 2 different patterns can be generated without any break. When we carefully look at table 6.1 below, we can easily notice that complementary patterns are lined up in inverted order. For example, first and last generated patterns are complementary patterns.

sno	week1	week2	week3	week4	week5
1	0	0	1	0	1
2	0	1	0	0	1
3	0	1	0	1	0
4	0	1	0	1	1
5	0	1	1	0	1
6	1	0	0	1	0
7	1	0	1	0	0
8	1	0	1	0	1
9	1	0	1	1	0
10	1	1	0	1	0

Table 6.1. All Feasible Patterns for a 6 Teamed League Having 1-Break at Most

6.2. Feasible Pattern Set Generation

Creating an appropriate schedule for a round-robin tournament with pattern assignment is a hard problem for sports scheduling. We need to select patterns, which can be completed into a schedule, for pattern set. Such a pattern set is named as **feasible pattern set**. Feasible pattern set choice is the key phase of the sports scheduling. Although the fact is that, exact characterization of the feasible pattern set is not known

yet (Miyashiro et. al, 2003). The problem of considering the pattern set whether it is feasible or not is defined as **pattern set feasibility problem**. Current Turkish League schedule's pattern set with n teams has some properties which are listed below. (n=18)

1. It consists of n/2 complementary pattern pairs to make the schedule available for n/2 pair of teams sharing the same stadium.
2. Total number of breaks of patterns is minimal including 2 patterns without breaks and n-2 patterns with 1-break.
3. Patterns can have breaks in all rounds excluding the first and last round of the season.

We divided this phase into 2 stages as well. The first one is generating a pattern set and the second one is feasibility check of the generated pattern set.

6.2.1. Pattern Set Generation

We have mentioned that current Turkish League Pattern Set has 3 basic characteristics. For providing these characteristics, following methodology including these characteristics is followed sequentially.

1st Step

We have 2n-2 generated candidate patterns to be chosen for a pattern set with n patterns. If we use Generate and Test algorithm to search for a feasible pattern set, we have different combinations of 2n-2 with n teams C (2n-2, n). When we look at the size of the search space, we can easily predict that this process will be exhaustive and will reduce performance. We illustrated before that generated complementary patterns are lined up in inverted order. The second half of the all patterns is the complementary patterns of the first half. Therefore, we can easily generate a pattern set including n/2 complementary pattern pairings by selecting n/2 patterns from first n-1 patterns of all patterns initially before adding selected patterns' complementary patterns, which is the second n-1 patterns of all patterns, to the initially selected patterns. Thus, we can reduce the search space to the C (n-1, n/2)

2nd Step

2 patterns of the n patterns don't have any breaks and n-2 patterns have 1 break to minimize total number of breaks for n teams. By fixing 2 patterns of the pattern set, n-2 patterns from 2n-4 candidate patterns having 1 break each. We provided this

characteristic by fixing one pattern to the pattern without break in the selection process of first n/2 pattern. Therefore, now we need to choose (n/2 -1) patterns from the first half of the candidate patterns excluding the fixed pattern. Thus, our search space is reduced to the C (n-2, n/2-1).

3rd Step

After pattern generation, we demand the rounds from the user in which patterns of pattern set are not allowed to have any breaks. Current Turkish League patterns don't have any break in the first round and last round of the season. 2 patterns in the first half of the candidate patterns are also eliminated from candidate patterns to construct rest of the pattern set. Thus, our search space is reduced to the C (n-4, n/2-1). This combination can be valid for scheduling leagues with n teams, if n is even and equal or greater than 6.

Problem 2: Generating pattern set considering these 3 characteristics.

Solution Methodology:

1st Step

Let p_k be the binary variable for the id of generated patterns ranging from p_1 to p_{2n-2}.

$$p_k = \begin{cases} 1, & \text{if pattern t is in the generated pattern set,} \\ 0 & \text{else} \end{cases}$$

Such that

$$\sum_{k=1}^{n-1} p_k = n/2 \qquad (6.3)$$

$$\sum_{k=1}^{n-1} p_k + p_{2n-2+1-k} = n \qquad (6.4)$$

Third set of constraint ensures that n/2 patterns are selected for pattern set from the first half of the generated patterns and the fourth set ensures that complementary patterns of initial selected n/2 patterns are selected to fulfill pattern set from the second half of the generated patterns. Table 6.2 shows all of the generated pattern sets that are not violating the constraints above, for a league with 6 teams. We can see that we have C (5, 3) number of pattern sets generated.

sno	pattern1	pattern2	pattern3	**pattern4**	pattern5	pattern6
1	1	2	3	8	9	10
2	1	2	4	7	9	10
3	1	2	5	6	9	10
4	1	3	4	7	8	10
5	1	3	5	6	8	10
6	1	4	5	6	7	10
7	2	3	4	7	8	9
8	2	3	5	6	8	9
9	2	4	5	6	7	9
10	3	4	5	6	7	8

Table 6.2. Generated Pattern Sets for a League with 6 Teams after Step 1

2nd Step

We are adding a new constraint to our solution and we generate pattern sets which are guaranteed to include patterns which do not have any breaks. The $(n/2)^{th}$ pattern of generated patterns is always the pattern without any break. Fifth constraint below provides this and the fourth constraint above ensures to include complementary pattern of the $(n/2)^{th}$ pattern.

$$p_{n/2} = 1 \qquad (6.5)$$

sno	pattern1	pattern2	pattern3	pattern4	pattern5	pattern6
1	1	2	3	8	9	10
4	1	3	4	7	8	10
5	1	3	5	6	8	10
7	2	3	4	7	8	9
8	2	3	5	6	8	9
10	3	4	5	6	7	8

Table 6.3. Generated Pattern Sets for a League with 6 Teams after Step 2

All of the pattern sets must include 3rd pattern. Size of the total generated pattern sets, which are not violating fifth constraint for a league with 6 teams, are decreased to C (4, 2) as shown in the Table 6.3.

3rd Step

We use y_t variable, which was defined before in the first phase of the scheduling. R is a set of rounds and patterns in pattern set can't have break in set R. The sixth constraint below ensures that total break minimization of selected patterns in which breaks not occurring in the rounds of set R.

$$\sum_{k=1}^{n-1} \sum_{t=1}^{n-2} p_k y_t = 2n\text{-}2 \quad \text{for all } t \notin R \qquad (6.6)$$

sno	pattern1	pattern2	pattern3	pattern4	pattern5	pattern6
8	2	3	5	6	8	9

Table 6.4. Generated Pattern Sets for a League with 6 Teams after Step 3

Table 6.4 shows all of the pattern sets having patterns in which breaks do not occur in the beginning and end of the season for a league with 6 teams. Our set R = {1, 4}. Thus, size of the generated pattern sets is decreased to C (2, 2) as shown in the table above.

6.2.2. Pattern Set Feasibility Check

In this stage of our program, we check the generated pattern set if it has a chance to be fit into a schedule. Initially, we check for some basic necessary conditions of feasible pattern set. Later, we use a heuristic in integer programming to eliminate some of the pattern sets whose chance of being completed into a schedule is less. Every feasible pattern set must satisfy the following two conditions (Nemhauser and Trick, 1998):

i. In each round, total numbers of 0's and 1's must be equal.

ii. All of the patterns of pattern set must be different.

Satisfaction of these two conditions may not be adequate. Miyashiro et. al (2003) proposed additional necessary conditions, which must be satisfied by every feasible pattern set, as listed below.

iii. Overall possible matches during tournament must be equal to C (n, 2) for a league with n teams.

iv. In a given pattern set, let *T* be an arbitrary subset of teams whose number is m. In each round, count the number of 1's and that of 0's in *T*, then take the minimum of the two. If the sum total of the minima is strictly less than C (m, 2), the pattern set is infeasible.

Although 4th condition is conjectured to be sufficient for a feasible pattern set, Briskorn (2008) stated that it was not proven yet and this condition's control can take long time. Thus, we proposed a new condition which is introduced below.

v. Our additional condition checks the each week's total number of match chances by calculating sum of the assigned week probability of matches of all pattern pairs.

Problem 3: Check for pattern sets if they are possible.

Solution Methodology:

We use p_k variable, which was defined before, to determine if pattern k is in the generated pattern set. Let *T* be the set of rounds. Let h_{kt} is the variable to determine if pattern k plays home match in round t and a_{kt} is the variable to determine if pattern k plays home match in round t (Nemhauser and Trick, 1998).

$$h_{kt} = \begin{cases} 1, & \text{if pattern k plays home match in round t,} \\ 0 & \text{Else} \end{cases}$$

$$a_{kt} = \begin{cases} 1, & \text{if pattern k plays away match in round t,} \\ 0 & \text{Else} \end{cases}$$

Such that

$$\sum_{k=1}^{2n-2} p_k (h_{kt}) = n/2 \qquad \text{for all } t \in T \qquad (6.7)$$

$$\sum_{k=1}^{2n-2} p_k (a_{kt}) = n/2 \qquad \text{for all } t \in T \qquad (6.8)$$

Seventh and eighth set of constraints ensures that total number of 0's and 1's of patterns in the pattern set are equal in each round. In our formula, we defined them as n/2 because league is organized with n, which is even in our program.

$$\sum_{t=1}^{n-1} | h_{kt} - h_{rt} | < 0 \qquad \text{for all (k, r), } k < r, \; r < 2n-2 \qquad (6.9)$$

Ninth set of constraints above shows that all of the generated patterns to generate pattern sets that are different.

sno	week1	week2	week3	week4	week5
2	0	1	0	0	1
3	0	1	0	1	0
5	0	1	1	0	1
6	1	0	0	1	0
8	1	0	1	0	1
9	1	0	1	1	0

sno	week1	week2	week3	week4	week5
sum	3	3	3	3	3

Table 6.5. Sample Generated Pattern Set and Summation of Patterns

In the previous section, we construct our pattern set with 6 patterns using complementary patterns. Table 6.5 shows our checks for constraints 7, 8 and 9 using pattern set example including 6 patterns (2, 3, 5, 6, 8, 9) which was generated before.

$$X_{krt} = \frac{| h_{kt} - h_{rt} |}{p_k \, p_r \sum_{s=1}^{n-1} | h_{ks} - h_{rs} |} \qquad \text{for all } p_k, p_r = 1, \; k < r$$

Let x_{krt} be the floating variable which shows the probability of match between pattern k and pattern r in round t and m_t is the total number of match chances in round t by calculating sum of the assigned week probability of matches of all pattern pairs. We can find this probability with the use of linear programming.

$$m_t = \sum_{k=1}^{2n-3} \sum_{r=k+1}^{2n-2} X_{krt} \qquad \text{for all } t \in T$$

$$\text{Max } (m_t) - \text{Min } (m_t) < 1.96 \qquad (6.10)$$

Tenth set of constraint ensures that the difference between weeks, whose total numbers of match chances are maximum and minimum, should be less than 1.96. This value is defined after the tests for leagues with 6, 8, 10, 12, 14, 16 and 18 teams. Pattern sets, which are checked by these constraints, are certainly being able to be completed into a schedule.

row	week	x	homepattern	awaypattern
1	3	1	2	5
2	3	1	6	9
3	4	0,5	8	9
4	5	0,5	8	9
5	4	0,5	2	3
6	5	0,5	2	3
7	1	0,5	3	6
8	2	0,5	3	6
9	1	0,5	5	8
10	2	0,5	5	8
11	1	0,333333333333	3	9
12	2	0,333333333333	3	9
13	3	0,333333333333	3	9

Table 6.6. Possibility of Assignment of Matches to the Rounds

Table 6.6 shows the match assignment possibilities to the week for some of the pattern pairs. For example, the match between patterns whose no is 5 and pattern whose no is 2 is restricted to be assigned to the round 3. So X_{253} is equal to 1 as shown in the 3rd column of the table. However X_{894} is equal to 0.5 because the match between pattern whose no is 9 and pattern whose no is 8 can be assigned to the round 4 and round 5. We have totally 45 assignment possibilities for the all of the pattern pairs in the pattern set. For example m_3 is calculated as below.

$$m_3 = X_{253} + X_{693} + X_{393} + \ldots + X_{293} = 1 + 1 + 0.33 + \ldots + 0.2 = 3.93$$

$$m_3 - m_1 = 3.93 - 2.77 = 1.16 < 1.96$$

Table 6.7 shows that maximum of these values is m_3 and minimum of these values is one of the other values. m_1 is chosen because of being first one. Calculation of the formula above proves that constraint 10 is not violated by this pattern set and this pattern set is feasible for other phases of our scheduling method.

m_1	m_2	m_3	m_4	m_5
2,77	2,77	**3,93**	2,77	2,77

Table 6.7. m_t Values of Sample Pattern Set

6.3. Constructing Timetable for Feasible Pattern Set

In this phase of our scheduling solution, we assign games to the rounds for a given pattern set. We use constraint programming method to construct a timetable. There are two objectives to be satisfied for the construction of 1RR tournament:

1. Every pattern in the pattern set must play every other pattern of pattern set.
2. Each pattern of pattern set must play one game in each round.

Problem 4: Construct a timetable for a given pattern set.

Solution Methodology:

Let set *P* be patterns of given pattern set and w_{ij} is the variable for the assigned round of the game between pattern whose no is *i* and pattern whose no is *j* and *i* is not equal to *j*. Let D_{ik} be the variable for the domain of w_{ik}. This problem can be formulated using the function which was introduced by Rasmussen (2006) below.

all_different (w_1, w_2... w_n): As the name suggests the constraint is satisfied when all the variables w_1. . . w_n are instantiated to different values.

We can adjust this function to our problem as the constraint 11 in addition to other constraints providing 1RR tournament for a given pattern set below.

$$\text{all\_different } (w_{i2}, w_{i3}, w_{i5}, w_{i6}, w_{i8}, w_{i9}) \quad \text{for all } i \in P \tag{6.11}$$

$$w_{ik} = 0 \quad \text{for all } i, k \in P, \; i = k \tag{6.12}$$

$$w_{ik} < n \quad \text{for all } i, k \in P \tag{6.13}$$

Firstly, we are using the variable ordering for w_{ik}, which was mentioned in 5.2.3. Due to high cost of dynamic ordering, we select static ordering to order variables. We sort the variables of pattern pairs in terms of their domain size.

row	size	homepattern	awaypattern
1	1	2	5
2	1	6	9
3	2	8	9
4	2	5	8
5	2	2	3
6	2	3	6
7	3	3	5
8	3	2	8
9	3	3	9
10	3	6	8
11	4	5	9
12	4	2	6
13	5	2	9
14	5	3	8
15	5	5	6

Table 6.8. Sorted Pattern Pairs for Given Pattern Set

We use branch and bound tree search technique in constraint programming with backjumping specification to assign rounds to games. For a given pattern set, games between pattern pairs will be assigned to rounds in order. This order is shown in Table 6.8 for a pattern set having 6 patterns (2, 3, 5, 6, 8 and 9).

$\mathbf{D_{25}}$ =	{3}	$\mathbf{D_{36}}$ =	{1, 2}	$\mathbf{D_{59}}$ =	{1, 2, 4, 5}
$\mathbf{D_{69}}$ =	{3}	$\mathbf{D_{35}}$ =	{3, 4, 5}	$\mathbf{D_{26}}$ =	{1, 2, 4, 5}
$\mathbf{D_{89}}$ =	{4, 5}	$\mathbf{D_{28}}$ =	{1, 2, 3}	$\mathbf{D_{29}}$ =	{1, 2, 3, 4, 5}
$\mathbf{D_{58}}$ =	{1, 2}	$\mathbf{D_{39}}$ =	{1, 2, 3}	$\mathbf{D_{38}}$ =	{1, 2, 3, 4, 5}
$\mathbf{D_{23}}$ =	{4, 5}	$\mathbf{D_{68}}$ =	{3, 4, 5}	$\mathbf{D_{56}}$ =	{1, 2, 3, 4, 5}

Table 6.9. Domains of Pattern Pairs for Given Pattern Set

Table 6.9 shows the initial domains of games of pattern pairs. Our root node is the assignment of w_{25} and our search for constructing timetable will proceed until round assignment of last node is completed.

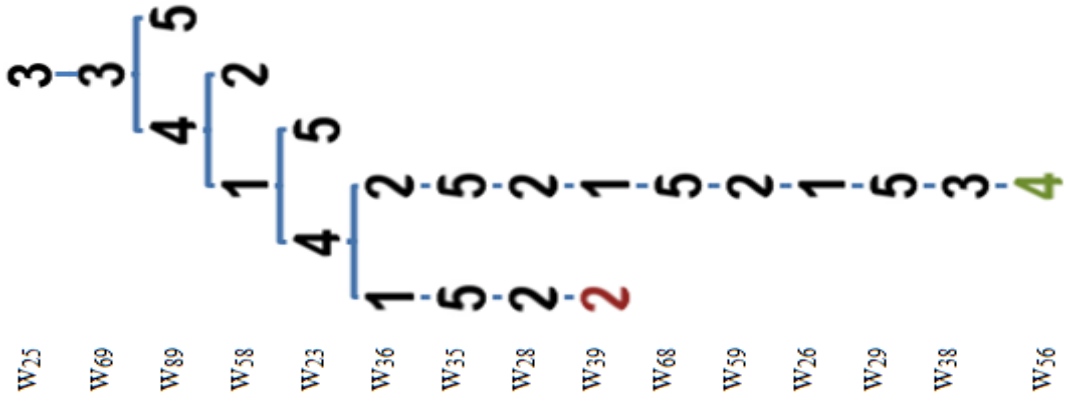


Figure 6.2. Tree Search B & B for Constructing Timetable

Figure 6.2 shows the branch and bound tree search method using the backjumping algorithm. After each node assignment, domains of rest of the variables are determined again related to the constraints (11 and 13). Arc consistency feature of constraint programming reduces the domains of the variables, which are not assigned after each node assignment. Before the assignment of w_{25}, domains of other games are shown in Table 6.9. After the assignment of w_{25}, variables which have changes of their domains are shown below.

$\mathbf{D_{35}} = \{4, 5\}$ $\mathbf{D_{28}} = \{1, 2\}$ $\mathbf{D_{29}} = \{1, 2, 4, 5\}$ $\mathbf{D_{56}} = \{1, 2, 4, 5\}$

After each node assignment, forward checking algorithm is used to detect possible future conflicts beforehand. In our program, our forward checking checks whether the domains of the unassigned variables are empty or not. If there is an empty domain, the branch fails. In Figure 6.2, red coloured node, which is an assignment of w_{39} shows the failing branch, which detects future conflict. After the assignment of w_{39}, domain of w_{59} becomes empty. If we don't use forward checking, we can't detect the failure until the search proceeds to the w_{59}. However, in our program, this conflict can be detected just after the assignment of w_{39}.

After detecting failure in the that branch of the node, backjumping algorithm provides jump over the nodes, whose assignment can't be the reason of conflict. For a given pattern set in our example, our search jumps from w_{39} to w_{35} by passing the node w_{28}. After the jumping to the w_{35}, D_{35} is checked if it has any alternative value for assignment of w_{35}. We can see in Figure 6.2 that there isn't any other option for the assignment of w_{35}. Thus, our search jumps back to the w_{36}. An alternative value, which is 2, for w_{36} is assigned. After the assignment of w_{36}, the search can proceed to the assignment of last game. After the assignment of last game, the search is finished and the given pattern set is completed into a timetable. Some pattern sets can be completed into more than one different timetables but in our algorithm, we didn't search any alternative timetables for a given pattern set to reduce the total search time. Because there is not a gurantee to be completed into a more than one different timetables for given pattern sets and after finding first timetable, search for alternative timetables may be timewasting process. 2^{nd} half schedule of the season can be easily deduced by reversing home and away patterns as shown in Table 6.10.

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

6	7	8	9	10
5-8	6-3	2-5	8-9	3-5
3-9	8-2	6-9	2-3	6-8
2-6	9-5	3-8	5-6	9-2

Table 6.10. Timetable for Given Pattern Set

6.4. Carry Over Effect Value Minimization

Carry Over Effect value and its importance were before mentioned in chapter 3. Constraint 14 below adds COE value minimization requirement to our scheduling problem. After construction of timetable, we calculate the COE value of timetable in this phase and then check its value if it is the best one.

Problem 5: Search for schedule with the minimized COE value.

Solution Methodology:

Let c_{ij} denotes the number of carry-over effects given by pattern i to pattern j in the schedule and let set P be patterns of the pattern set of the given schedule. Our problem's mathematical formula is shown as constraint 14. Our solution method solves this problem in 2 steps.

$$\textbf{Min} \sum_{i,\,j} (c_{ij})^2 \quad \text{for all i, j} \in P \text{ (Russell, 1980)} \qquad (6.14)$$

1st Step

Calculation of COE value for a timetable for a given pattern set.

T/W	1	2	3	4	5
2	6	8	5	3	9
3	9	6	8	2	5
5	8	9	2	6	3
6	2	3	9	5	8
8	5	2	3	9	6
9	3	5	6	8	2

(a)

j/i	2	3	5	6	8	9
2	0	0	1	0	3	1
3	3	0	1	1	0	0
5	1	1	0	1	1	1
6	1	0	1	0	0	3
8	0	1	1	3	0	0
9	0	3	1	0	1	0

(b)

Table 6.11. (a) Schedule for Given Pattern Set (b) COE Matrix (6x6) of Schedule

COE value of the schedule in Table 6.11 (a) is calculated as below using formula which was proposed by Russell (1980).

$$\sum_{i,j} (c_{ij})^2 = 5*(3)^2 + 15*(1)^2 + 16*(0)^2 = 60$$

After each schedule generation, COE value of the new generated schedule is calculated.

2nd Step:

Check for the calculated COE value if it is minimum one.

When the first schedule of the program is generated, COE value of this schedule is tagged as best-known value. In addition to this, generated schedule is tagged as best schedule and stored.

After each schedule generation, calculated COE value for the new generated schedule is compared with best-known value. If the COE value of new generated schedule is lower than the best-known value, the new value is tagged as best-known value and new generated schedule is exchanged with current best schedule.

We have only one feasible pattern set to be scheduled for 6 teams. Thus, Table 6.12 shows the pattern sets which can be completed into a schedule. The second column shows the COE value of schedule for a pattern set having patterns ranging from pt1 to pt10 in the table. After whole over the feasible pattern sets, schedule of pattern set, whose COE value is 208 is selected as most suitable schedule for 10 teams meeting the constraints. Third column shows the consumed time for generating timetable for relative pattern set.

sno	coe_	time	pt1	pt2	pt3	pt4	pt5	pt6	pt7	pt8	pt9	pt10
2	260	00:00:16	2	3	4	5	9	10	14	15	16	17
7	208	00:00:05	2	4	5	7	9	10	12	14	15	17
12	226	00:00:05	3	4	5	8	9	10	11	14	15	16
14	210	00:00:05	4	5	7	8	9	10	11	12	14	15

Table 6.12. Scheduled Pattern Sets with Relative COE Values

Important Note: It is important to remind that a pattern set can be completed into more than one different schedule as we mentioned before in 6.3. Consequently COE value of them can be different. For example, although schedule of current Turkish League is 3876, our program can find a schedule whose COE value is 1232 for the same pattern set of current Turkish League and % 68 improvements is achieved in COE value of that pattern set.

6.5. Assignment of Teams to the Patterns

Once the schedule is generated, the final computational phase is assigning the teams to the given patterns. In this phase, individualized constraints of teams must be taken into account. These can include stadium scheduling conflicts, rivalry week requirements, police requirements etc. In our method, we take stadium conflicts into account and assignments are made by considering this constraint.

Problem 6: Assignments of teams to the patterns.

Solution Methodology:

Let set P be patterns of the pattern set of the given schedule and let t_i is a variable for the team i's pattern ranging from 1 to n, which is the number of teams.

$$\text{all\_different}\ (t_1, t_2 \ldots t_i) \qquad \text{for all } t_i \in P,\ i<n+1 \qquad (6.15)$$

Let (i, j) be the pair of teams which should be assigned to complementary patterns and let D_i is the domain for team i's pattern assignment.

$$t_1 + t_j = 2n-1 \qquad \text{for all } t_i, tj \in P \qquad (6.16)$$

15th constraint above shows that each team must be assigned to different pattern of a given pattern set and 16th constraint ensures that team i and team j have complementary patterns. We use constraint programming branch and bound method to solve this problem like the solution of the scheduling problem in 6.3.

Firstly, we require from user the team pairs, which should have complementary patterns. Then, we are using the variable ordering for t_i, which was mentioned in 5.2.3. Due to we select static ordering to order variables. We sorted teams in terms of their domain size. If domain sizes of teams are equal, they are sorted in terms of their team id ascending.

All of the domains of the teams for a given pattern set are shown in the Table 6.13 below.

D_1 =	{2, 3, 5, 6, 8, 9}	**D_4 =**	{2, 3, 5, 6, 8, 9}
D_2 =	{2, 3, 5, 6, 8, 9}	**D_5 =**	{2, 3, 5, 6, 8, 9}
D_3 =	{2, 3, 5, 6, 8, 9}	**D_6 =**	{2, 3, 5, 6, 8, 9}

Table 6.13. Initial Domains of the Teams

Figure 6.3 shows the branch-bound algorithm in constraint programming for the assignment of teams to the patterns. For the schedule of this figure (t_3, t_4) and (t_5, t_6).

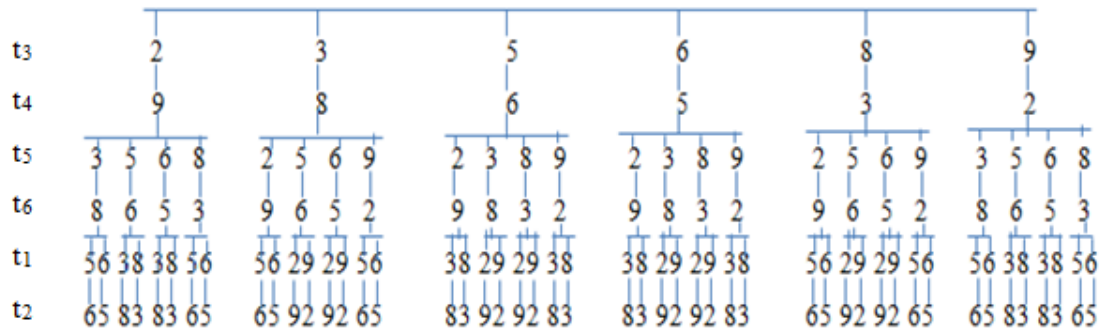


Figure 6.3. Tree Search Solution Sample for the Pattern Assignment of Teams

Teams are randomly assigned to the one of the patterns, which is in the domain of that team. Arc consistency feature of constraint programming reduce the domains of the variables, which are not assigned, after each node assignment. Before the

assignment of t_3, domains of other games are shown in Table 6.13. For example t_3 is randomly assigned to the pattern 5. After the assignment of t_3, variables which have changes of their domains are shown below in the Table 6.14.

D_1 =	{2, 3, 8, 9}	**D_3 =**	**assigned**	**D_5 =**	{2, 3, 8, 9}
D_2 =	{2, 3, 8, 9}	**D_4 =**	{6}	**D_6 =**	{2, 3, 8, 9}

Table 6.14. Domains of the Teams after First Assignment

After each team's assignment of the pattern, Table 6.15 (a) shows the assigned patterns of teams and Table 6.15 (b) shows the first half of the arranged fixture.

SNO	Name	Pattern
1	Fenerbahçe	2
2	Galatasaray	9
3	Beşiktaş	3
4	Bursaspor	8
5	Kasımpaşa	6
6	Gençlerbirliği	5

(a)

week	hometeam	awayteam
1	Kasımpaşa	Fenerbahçe
1	Galatasaray	Beşiktaş
1	Bursaspor	Gençlerbirliği
2	Beşiktaş	Kasımpaşa
2	Fenerbahçe	Bursaspor
2	Gençlerbirliği	Galatasaray
3	Galatasaray	Kasımpaşa
3	Gençlerbirliği	Fenerbahçe
3	Bursaspor	Beşiktaş
4	Beşiktaş	Fenerbahçe
4	Kasımpaşa	Gençlerbirliği
4	Galatasaray	Bursaspor
5	Fenerbahçe	Galatasaray
5	Bursaspor	Kasımpaşa
5	Gençlerbirliği	Beşiktaş

(b)

Table 6.15. (a) Assigned Patterns to the Teams (b) First Half Fixture of the Season

CHAPTER 7

COMPUTATIONAL RESULTS

In this chapter, results of our program are presented for the 2RR league with n teams, when n is even. We used C# programming language and MSSQL database language to code our project. Table 7.1 shows the results of the application. 1st column shows the minimum COE value of schedules alternatives, which are generated by our program. 2nd column shows the total number of breaks of the schedule which has minimum COE value. 3rd column shows the total number of pattern sets generated by phase 1. 4th column is the number of feasible pattern sets, which are defined in phase 2. 5th column is the number of the pattern sets which can be completed into a schedule in phase and last column shows total amount of time to finish our program's search process. Our tests are performed using the computer with Core I7 1.6 GHz.

# of Teams	COE Value (Min)	#of Breaks	# of PSs	# of Feasible PSs	# of Scheduled PSs	Total Time
6	60	12	1	1	1	1 sec.
8	104	18	4	2	2	7 sec.
10	208	24	15	4	4	23 sec.
12	316	30	56	6	6	1,5 min.
14	498	36	210	9	9	6 min.
16	816	42	792	14	14	10 min.
18	948	48	3003	19	19	7,5 hours

Table 7.1. Computational Results

The tests are performed with pattern sets having patterns which have no break in first and last weeks of the season. Total number of breaks is calculated for the whole season; however COE value is calculated for 1RR tournament. Figure 7.1 is the graph in which blue bars show the COE values of schedules for leagues of different number of teams using current Turkish League scheduling system which is in canonical way and

red bars show the COE values of schedules for leagues of different number of teams using our study and green line shows the improvement percentage of COE values for leagues of different number of teams.

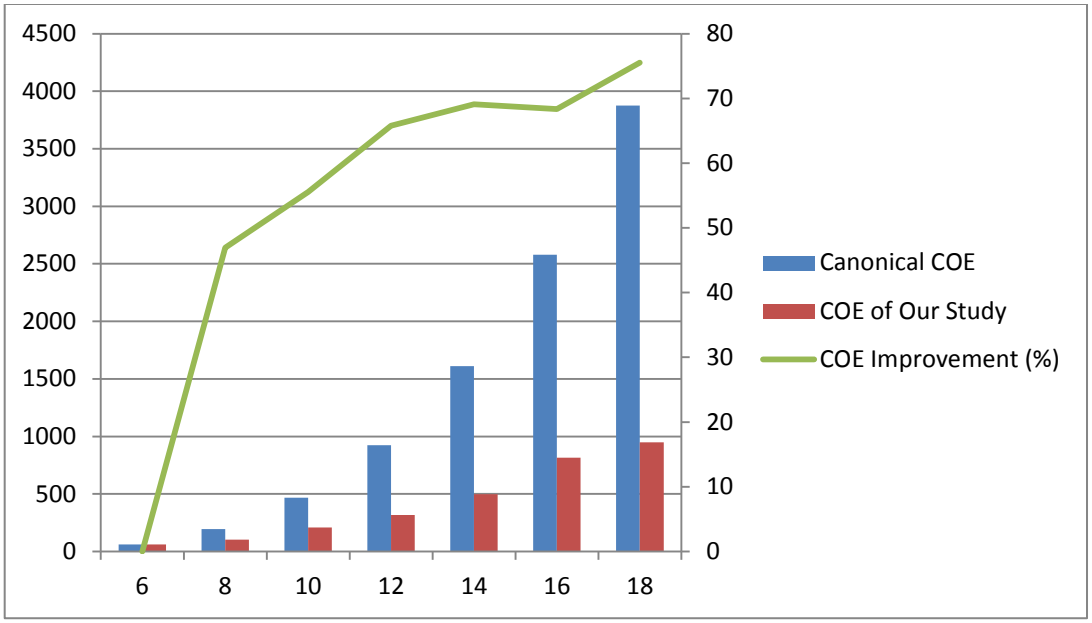


Figure 7.1. COE Value Comparison of Canonical Method and Our Study

Scheduling phase for a feasible pattern set is always achieved in the first branch of the tree for the league with 2^a teams (2, 8, 16). Lasting time of scheduling phase for pattern sets having 18 patterns can vary from 1 minute to 70 minutes. Figure 7.2 shows the scheduling time performances of our program for leagues of different number of teams. Blue line in this figure is the total scheduling time of our program. During this time more than 1 schedule are generated as shown in the 5^{th} column of Table 7.1. Red line in this figure is the average time per schedule which increases as the number of teams increases.

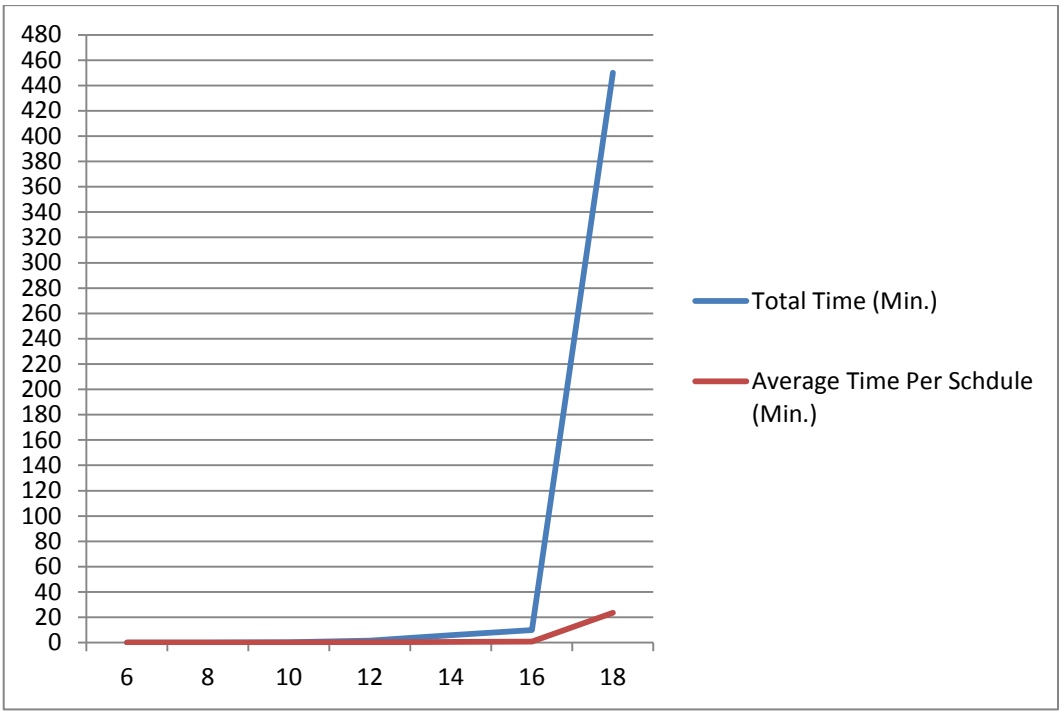


Figure 7.2. Scheduling Time Performances

We can see in Table 7.1 that, all feasible pattern sets can be completed into a schedule (% 100 feasible pattern set selection success) for leagues with 6, 8, 10, 12, 14, 16 and 18 teams. It can be succeeded by our new proposed feasible pattern set selection constraint which is detailed in section 6.2 as constraint 10 (equation 6.10). Although, there are more amount of pattern sets which may be completed into a schedule, it is a hard task to select feasible pattern set and our program can succeed this task thanks to constraint 10 (equation 6.10). In this constraint, we found the common value as 1.96 which provides %100 feasible pattern set selection success ratio for all leagues having different number of teams after many trials. When this value is increased, although more schedules may be generated for COE value selection, feasible pattern set selection ratio decreases and this decrease causes huge increase in lasting time of the program to be completed. Because lasting time of scheduling phase attempts for a pattern set which can't be completed into a schedule may increase to 2 hours.

CHAPTER 8

CONCLUSION

Turkish Soccer League is becoming one of the most important sport competitions by increasing its economical worth and effectiveness of its schedule to meet requirements from different stakeholders should be strengthened as well. Although many constraints may be taken into account primarily, we thought that fairness of schedule should be provided initially because of ongoing match-fixing case and rumors of it. So, we conducted this study to meet two important fairness criterions of schedule which are minimizing total number of break and minimizing carry over effect value, in addition to must requirements such as prevention of conflicting stadiums.

We discussed that there are many approaches and algorithms which can be applicable for the large-scale soccer scheduling problems. We chose the variant of *first-break-then-schedule* approach, which was proposed by Rasmussen and Trick (2008), by decomposing our problem into sub problems. For Turkish Soccer League, this approach seems the most suitable one because consideration of break minimization and conflicting stadiums are the break-related constraints which are already met in the current application of Turkish Soccer League's scheduling method which is canonical.

Minimization of break and COE value are succeeded using integer programming, however phases, which are scheduling and assignment of patterns, are done using constraint programming. We proposed a new constraint (constraint 10 as equation 6.10) to improve feasible pattern set selection ratio. Total number of the defined constraints in our paper is 16 but our method is open to be improved to meet more constraints. Improving solution process of our method, finding a schedule with decreased COE value and addition of new constraints from various stakeholders is the issue which can be explored in future work. These include:

1. If the solution process can be accelerated by using professional optimization libraries, 3rd phase searching of a first schedule for a given pattern, can be changed and search can be extended to find more than 1 schedule for a given pattern set.
2. If individual constraints increase and conflicts of these constraints are emerged, phase 5 assigning of patterns to the teams can be done using integer

programming by aiming the minimization of total violation of constraints, which have the weighted importance coefficient.

As we have seen the points above, there can be many possible extensions to our solution method.

In conclusion sport scheduling becomes an important research area in other leagues and there is a trend to cancel canonical scheduling method in other countries where computer sciences present more advanced techniques. Since the amount of money in sports leagues increase, constraints from different stakeholders will increase and there will be always a lot to do for the optimality of sports scheduling.

REFERENCES

Anderson, I. 1999. Balancing Carry-Over Effects in Tournaments. *In F. Holroyd, K. Quinn, C. Rowley, and B. Webb, editors, Combinatorial Designs and Their Applications, CRC Research Notes in Mathematics :* 1–16.

Anson, S. and S. Lester. 2007. Sports Scheduling: Algorithms and Applications.

Barták, R. 1995. Constraint Propagation and Backtracking-Based Search.

Barták, R. 1998. Online Guide to Constraint Programming. http://kti.mff.cuni.cz/~bartak/constraints/ (accessed June 11, 2013)

Bartsch, T., A. Drexl, S. Kröger. 2006. Scheduling the Professional Soccer Leagues of Austria and Germany. *Computers and Operations Research* 33: 1907-1937.

Benders, J.F. 1962. Partitioning Procedures for Solving Mixed-Variables Programming Problems. *Numerische Mathematik* 4: 238–252.

Briskorn, D. 2008. Feasibility of Home-Away-Pattern Sets for Round Robin Tournaments. *Operations Research Letters* 36: 283-284.

Brouwer, A.E., G. Post, G.J. Woeginger. 2008. Tight Bounds for Break Minimization. *Journal of Combinatorial Theory (A)* 115: 1065–1068.

Chun, H.W., N.M. Lam. 2002. Using Heuristics in Constraint-Based Sports Tournament Timetabling. *The 6th World Multiconference on Systemics, Cybernetics and Informatics*.

Country Coefficients 2012/13, UEFA. 2013. http://www.uefa.com/memberassociations/uefarankings/country/index.html (accessed June 11, 2013).

Cutting Plane Techniques for Getting Improved Bounds. http://www.columbia.edu/~cs2035/courses/ieor4600.S07/cuttingplanemit.pdf (accessed June 11, 2013).

de Werra, D. 1980. Geography, Games and Graphs. *Discrete Applied Mathematics* 2: 327–337.

de Werra, D. 1981. Scheduling in Sports. *In P. Hansen, editor, Studies on Graphs and Discrete Programming*: 381–395.

de Werra, D. 1982. Minimizing Irregularities in Sports Schedules Using Graph Theory. *Discrete Applied Mathematics* 4: 217–226.

de Werra, D. 1988. Some Models of Graphs for Scheduling Sports Competitions. *Discrete Applied Mathematics* 21: 47–65.

Della Croce, F. and D. Oliveri. 2006. Scheduling the Italian Football League: an ILP-Based Approach. *Computers and Operations Research* 33: 1963-1974.

Duran, G., M. Guajardo, J. Miranda, D. Saure, S. Souyris, A. Weintraub, R. Wolf. 2007. Scheduling the Chilean Soccer League by Integer Programming. *Interfaces* 37: 539-552.

Easton K., G. Nemhauser, M. Trick. 2004. Sports Scheduling. *in Handbook of Scheduling: Algorithms, Models and Performance Analysis, (J.T. Leung, ed.):* 52.1-52.19.

Fiallos, J., J. Perez, F. Sabillon, M. Licona. 2010. Scheduling Soccer League of Honduras Using Integer Programming. *Proceedings of the 2010 Industrial Engineering Research Conference*.

Gaschnig, J. 1979. Performance Measurement and Analysis of Certain Search Algorithms. *Technical Report CMU-CS*: 79-124.

Goossens, D. and F.C.R. Spieksma. 2009. Scheduling the Belgian Soccer League. *Interfaces* 39: 109-118.

Goossens, D. and F.C.R. Spieksma. 2012. Soccer Schedules in Europe: An Overview. *Journal of Scheduling* 15: 641-651.

Guedes, A. and C.C. Ribeiro. 2009. A Hybrid Heuristic for Minimizing Weighted Carry-Over Effects in Round Robin Tournaments. *Proceedings of the 4th*

Multidisciplinary International Conference on Scheduling: Theory and Applications (MISTA'09).

Guedes, A. and C.C. Ribeiro. 2011. A Heuristic for Minimizing Weighted Carry-Over Effects in Round Robin Tournaments. *Journal of Scheduling* 14: 655-667.

Hamiez, J.P. and J.K. Hao. 2006. Sports League Scheduling: Enumerative Search for prob026 from CSPLib. *F.Benhamou (ed.): CP 2006, Lecture Notes in Computer Science 4204, Springer*: 716-720.

Haralick, R.M. and G.L. Elliott. 1980. Increasing Tree Search Efficiency for Constraint Satisfaction Problems. *Artificial Intelligence* 14: 263-313.

Henz, M., T. Müller, S. Thiel. 2004. Global Constraints for Round Robin Tournament Scheduling. *European Journal of Operational Research* 153: 92-101.

Kendall, G. 2008. Scheduling English Football Fixtures over Holiday Periods. *Journal of the Operational Research Society* 59: 743-755.

Kendall, G., S. Knust, C.C. Ribeiro, S. Urrutia, 2010. Scheduling in Sports: An Annotated Bibliography. *Computers and Operations Research* 37: 1-19.

Kidd, M.P. 2010. A Tabu-Search for Minimising the Carry-Over Effects Value of a Round-Robin Tournament.

Kirkman, T. 1847. On a Problem in Combinations. *Cambridge Dublin Math Journal* 2: 191–204.

Meleke, U. July 19, 2012. Fikstür Çekimi Adil miydi?. *Milliyet Gazetesi, 19 July 2012*. spor.milliyet.com.tr/fikstur-cekimi-adil-miydi /spor/sporyazardetay/19.07.2012/1568955/default.htm (accessed June 11, 2013).

Meleke, U. July 25, 2010. Şu Fikstür Meselesi. *Milliyet Gazetesi, 25 July 2010*. http://www.meleke.com/?p=2467 (accessed June 11, 2013).

Meleke, U. April 23, 2009. Fikstür Avantajı Kimde?. *Milliyet Gazetesi, 23 April 2009*. http://www.meleke.com/?p=192 (accessed June 11, 2013).

Meleke, U. August 25, 2008. Utanç Vesikası Lig Fikstürü. *Milliyet Gazetesi, 25 August 2008*. http://www.meleke.com/?p=58 (accessed June 11, 2013).

Mitchell, J. 2001. Branch and Cut Algorithms for Integer Programming. *Encyclopedia of Optimization* II: 519-525.

Miyashiro, R., H. Iwasaki, T. Matsui. 2003. Characterizing Feasible Pattern Sets with a Minimum Number of Breaks. E. *Burke and P. De Causmaecker (eds.), PATAT 2002, Lecture Notes in Computer Science 2740, Springer*: 78-99.

Miyashiro, R. and T. Matsui. 2003. Round-Robin Tournaments with a Small Number of Breaks.

Miyashiro R. and T. Matsui. 2006. Minimizing the Carry-Over Effects Value in a Round-Robin Tournament. *E. Burke and H. Rudova (eds.), PATAT 2006, Proceedings*: 402-405.

Nemhauser, G.L., M.A. Trick. 1998. Scheduling a Major College Basketball Conference. *Operations Research* 46: 1-8.

Nurmi, K., D. Goossens, T. Bartsch, F. Bonomo, D. Briskorn, G. Duran, J. Kyngas, J. Marenco, C.C. Ribeiro, F. Spieksma, S. Urrutia, R. Wolf. 2010. A Framework for Scheduling Professional Sport Leagues. *IAENG TRANSACTIONS ON ENGINEERING TECHNOLOGIES VOLUME* 5: *Special Edition of the International MultiConference of Engineers and Computer Scientists 2009. AIP Conference Proceedings, Volume* 1285: 14-28.

Post, G., G.J. Woeginger. 2006. Sports Tournaments, Home-Away Assignments, and the Break Minimization Problem. *Discrete Optimization* 3:165–173.

Rasmussen, R.V. 2006. Hybrid IP/CP Methods for Solving Sports Scheduling Problems.

Rasmussen, R.V. and M.A. Trick. 2007. A Benders Approach for the Constrained Minimum Break Problem. *European Journal of Operational Research* 177:198–213.

Rasmussen, R.V. 2008. Scheduling a Triple Round Robin Tournament for the Best Danish Soccer League. *European Journal of Operational Research 185*: 795-810.

Rasmussen, R.V. and M.A. Trick. 2008. Round Robin Scheduling - A Survey. *European Journal of Operational Research* 188: 617-636.

Recalde, D., R. Torres, P. Vaca. 2012. Scheduling the Professional Ecuadorian Football League by Integer Programming.

Regin, J.C. 2001. Minimization of the Number of Breaks in Sports Scheduling Problems Using Constraint Programming. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science* 57: 115-130.

Ribeiro, C.C. and S. Urrutia. 2007. Scheduling the Brazilian Soccer Tournament with Fairness and Broadcast Objectives. *Lecture Notes in Computer Science 3867, Springer*: 149-159.

Ribeiro, C.C. and S. Urrutia. 2009. Bicriteria Integer Programming Approach for Scheduling the Brazilian National Soccer Tournament. *Proceedings of the Third International Conference on Management Science and Engineering Management*: 46–49.

Ribeiro, C.C. and S. Urrutia. 2010. Soccer Scheduling Goaaaaal!.

Ribeiro, C.C., S. Urrutia. 2011. Scheduling the Brazilian Soccer Tournament: Solution Approach and Practice. *Interfaces*.

Ribeiro, C.C. 2012. Sports Scheduling: Problems and Applications. *International Transactions in Operational Research 19*: 201-226.

Rossi, F., P. van Beek, T. Walsh. 2008. Constraint Programming. *Handbook of Knowledge Representation*: 181-211.

Russell, K.G. 1980. Balancing Carry-Over Effects in Round Robin Tournaments. *Biometrika* 67: 127-131.

Solving Integer Programming with Branch and Bound Technique. http://www.columbia.edu/~cs2035/courses/ieor4600.S07/bb-lecb.pdf (accessed June 11, 2013).

Sports Business Group. 2012. *Deloitte Annual Review of Football Finance 2012*.

Trick, M.A. 2001. A Schedule-Then-Break Approach to Sports Timetabling. *E. Burke and W. Erben (eds.), PATAT 2000, Lecture Notes in Computer Science 2079, Springer*: 242-252.

Trick, M.A. 2003. Integer and Constraint Programming Approaches for Round-Robin Tournament Scheduling. *E. Burke and P. De Causmaecker (eds.), PATAT 2002, Lecture Notes in Computer Science 2740, Springer*: 63-77.

Trick, M.A. 2004. Using Sports Scheduling to Teach Integer Programming. *INFORMS Transactions on Education* 5: 10-17.

Wikipedia. Backjumping. http://en.wikipedia.org/wiki/Backjumping (accessed June 11, 2013).