

M.SC. ENGG. THESIS

# An Improved Round Robin Tournament Ranking Algorithm

by

Afsana Ahmed Munia

Submitted to

Department of Computer Science and Engineering

in partial fulfillment of the requirements for the degree of  
Master of Science in Computer Science and Engineering



Department of Computer Science and Engineering

Bangladesh University of Engineering and Technology (BUET)

Dhaka 1000

March 04, 2017

The thesis titled “An Improved Round Robin Tournament Ranking Algorithm”, submitted by Afsana Ahmed Munia, Student Number: 0413052032 P, Session April 2013, to the Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, has been accepted as satisfactory in partial fulfillment of the requirements for the degree of Master of Science in Computer Science and Engineering and approved as to its style and contents. The examination held on March 04, 2017.

## Board of Examiners

1. 

Dr. M. Kaykobad  
Professor

Department of Computer Science and Engineering, BUET, Dhaka.


Chairman  
(Supervisor)

2. 

Dr. Md. Sohel Rahman  
Professor and Head

Department of Computer Science and Engineering, BUET, Dhaka

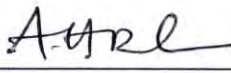
Member  
(Ex-Officio)

3. 

Dr. Muhammad Abdullah Adnan  
Assistant Professor

Department of Computer Science and Engineering, BUET, Dhaka

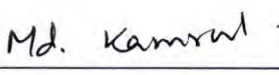
Member

4. 

Dr. Atif Hasan Rahman  
Assistant Professor

Department of Computer Science and Engineering, BUET, Dhaka

Member

5.  04/03/2017

Dr. Md. Kamrul Hasan  
Associate Professor

Department of Computer Science and Engineering, Islamic University  
of Technology, Board Bazar, Gazipur, Bangladesh.

Member  
(External)

Dedicated to my loving parents and husband

# Candidate's Declaration

This is hereby declared that the work titled "An Improved Round Robin Tournament Ranking Algorithm" is the outcome of research carried out by me under the supervision of Dr. M. Kaykobad, in the Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, Dhaka 1000. It is also declared that this thesis or any part of it has not been submitted elsewhere for the award of any degree or diploma.



---

Afsana Ahmed Munia

Candidate

# Acknowledgment

I would like to express my sincere gratitude to my supervisor Prof. Dr. M Kaykobad for the continuous support of my M.Sc. study and research, for his patience, motivation, enthusiasm, and immense knowledge. His guidance helped me all along this research as well as writing this thesis. He has always pointed me in the right direction when I was lost and supported me when I was on the right path. I could not have imagined having a better supervisor and mentor for my M.Sc. study.

Besides my supervisor, I would like to thank the rest of the exam committee: Dr. Md. Sohel Rahman, Dr. Muhammad Abdullah Adnan, Dr. Atif Hasan Rahman and Dr. Md. Kamrul Hasan for their encouragement and insightful comments.

In this regard, I remain ever grateful to my beloved parents, who always exists as sources of inspiration behind every success of mine I have ever made.

# Abstract

A round-robin tournament (or all-play-all tournament) is a competition “in which each contestant plays all other contestants in turn”. The problem of ranking players in a round-robin tournament with a win or a loss outcome of every match is to rank players according to their performances in the tournament. This tournament structure also arises in other environments, for example, in the problems of soliciting customer preferences regarding a set of products, establishing funding priorities of a set of projects, establishing searching priorities for a set of search engines in the Internet. In this thesis, we have improved previously developed MST (Majority Spanning Tree) algorithm for solving this problem, where the number of violations has been chosen as the criterion of optimality. Whereas in previous MST algorithms a subset A of consecutively ranked players could be swapped with a subset B of consecutively ranked players, where A and B are consecutively ranked, in this improved version we have allowed A and B to be any disjoint subset. We also developed another Hybrid algorithm using five related algorithms (Sort, Hamiltonian path, Arrange, MST and Improved MST) and compare the performance of these algorithms for the same sets of input data. Experimental results suggest that these new algorithm outperforms the existing ones.

# Contents

Board of Examiners	ii
Dedication	iv
Candidate's Declaration	v
Acknowledgment	vi
Abstract	vii
1 Introduction	1
1.1 The Round Robin Tournament Problem .....	3
1.2 Literature Survey .....	4
1.2.1 Time Relaxed Single Round Robin Tournament Problem (TRSRR)	7
1.2.2 Time Relaxed Double Round Robin Tournament (TRDRRT).....	7
1.3 Objective of the Thesis .....	8
1.4 Organization of the Thesis .....	8
2 Existing Algorithms	9
2.1 Iterated Kendall Algorithm.....	9
2.2 Generalized Iterated Kendall Algorithm.....	11



2.3	Hamiltonian Path Algorithm.....	13
2.4	Arrange Algorithm .....	15
2.5	The Feedback Arc Set Problem .....	16
2.5.1	Some Applications of FAS.....	17
2.6	Sorting Algorithms.....	18
2.7	A PTAS for Weighted Feedback Arc Set on Tournaments .....	19
3	Majority Spanning Tree	20
3.1	Introduction to MST .....	20
3.2	Existing MST Algorithm For Round Robin Tournament Problem .....	22
3.2.1	Majority Spanning Tree Algorithm .....	22
3.2.2	A Modified Algorithm For Round-Robin Tournament .....	25
3.3	Limitations in Existing Algorithms.....	28
3.4	Improved MST Algorithm .....	28
3.4.1	Procedure of Improved MST.....	30
3.5	Hybrid Algorithm.....	32
4	Implementations and Experiments	35
4.1	Implementation .....	35
4.1.1	Implementation of Sort .....	35
4.1.2	Implementation of HP.....	36
4.1.3	Implementation of Arrange .....	36
4.1.4	Implementation of MST .....	36
4.1.5	Implementation of Improved MST .....	37
4.2	Experiments.....	37

4.2.1	Graphical Analysis of the Results .....	44
5	Conclusion	47
5.1	Contribution .....	47
5.2	Future Works .....	48
References		

# List of Figures

1.1	Representation of results of a Round-Robin Tournament in a digraph . . .	4
2.1	Procedure of Iterated Kendall algorithm .....	10
2.2	Before applying Hamiltonian Path algorithm .....	13
2.3	After applying Hamiltonian Path algorithm .....	13
3.1	Example of fundamental cutset.....	21
3.2	Majority Spanning Tree.....	22
3.3	A Tournament digraph and a matrix of the result of the tournament.....	30
3.4	The algorithm divides the graph into three sets.....	30
3.5	Situation of digraph after swapping set I and set K.....	31
3.6	Situation of digraph after swapping set I and set J.....	32
3.7	Situation of digraph after swapping set J and set K .....	32
4.1	Comparison among MST, Improved MST and Hybrid Algorithm.....	44
4.2	Situation in worst case .....	45
4.3	Situation in best case .....	45
4.4	Average computational time required by each algorithm.....	46

# List of Tables

1.1	Four team Round Robin Tournament.....	2
1.2	Five team Round Robin Tournament .....	2
1.3	Representation of the game result.....	4
1.4	Time Relaxed Single Round Robin Tournament .....	7
4.1	Number of upsets for different algorithms .....	38
4.2	Average number of upsets for different algorithms .....	42
4.3	Average computational time of three algorithms in seconds .....	43
4.4	Ranking of various algorithms for 10 players.....	44



M.SC. ENGG. THESIS

# An Improved Round Robin Tournament Ranking Algorithm

by

Afsana Ahmed Munia

Submitted to

Department of Computer Science and Engineering

in partial fulfillment of the requirements for the degree of  
Master of Science in Computer Science and Engineering



Department of Computer Science and Engineering

Bangladesh University of Engineering and Technology (BUET)

Dhaka 1000

March 04, 2017

The thesis titled “An Improved Round Robin Tournament Ranking Algorithm”, submitted by Afsana Ahmed Munia, Student Number: 0413052032 P, Session April 2013, to the Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, has been accepted as satisfactory in partial fulfillment of the requirements for the degree of Master of Science in Computer Science and Engineering and approved as to its style and contents. The examination held on March 04, 2017.

## Board of Examiners

1. 

Dr. M. Kaykobad  
Professor

Department of Computer Science and Engineering, BUET, Dhaka.

Chairman  
(Supervisor)

2. 

Dr. Md. Sohel Rahman  
Professor and Head

Department of Computer Science and Engineering, BUET, Dhaka

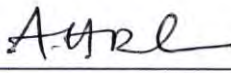
Member  
(Ex-Officio)

3. 

Dr. Muhammad Abdullah Adnan  
Assistant Professor

Department of Computer Science and Engineering, BUET, Dhaka

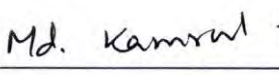
Member

4. 

Dr. Atif Hasan Rahman  
Assistant Professor

Department of Computer Science and Engineering, BUET, Dhaka

Member

5.  04/03/2017

Dr. Md. Kamrul Hasan  
Associate Professor

Department of Computer Science and Engineering, Islamic University  
of Technology, Board Bazar, Gazipur, Bangladesh.

Member  
(External)



Dedicated to my loving parents and husband

# Candidate's Declaration

This is hereby declared that the work titled "An Improved Round Robin Tournament Ranking Algorithm" is the outcome of research carried out by me under the supervision of Dr. M. Kaykobad, in the Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, Dhaka 1000. It is also declared that this thesis or any part of it has not been submitted elsewhere for the award of any degree or diploma.



---

Afsana Ahmed Munia

Candidate

# Acknowledgment

I would like to express my sincere gratitude to my supervisor Prof. Dr. M Kaykobad for the continuous support of my M.Sc. study and research, for his patience, motivation, enthusiasm, and immense knowledge. His guidance helped me all along this research as well as writing this thesis. He has always pointed me in the right direction when I was lost and supported me when I was on the right path. I could not have imagined having a better supervisor and mentor for my M.Sc. study.

Besides my supervisor, I would like to thank the rest of the exam committee: Dr. Md. Sohel Rahman, Dr. Muhammad Abdullah Adnan, Dr. Atif Hasan Rahman and Dr. Md. Kamrul Hasan for their encouragement and insightful comments.

In this regard, I remain ever grateful to my beloved parents, who always exists as sources of inspiration behind every success of mine I have ever made.

# Abstract

A round-robin tournament (or all-play-all tournament) is a competition “in which each contestant plays all other contestants in turn”. The problem of ranking players in a round-robin tournament with a win or a loss outcome of every match is to rank players according to their performances in the tournament. This tournament structure also arises in other environments, for example, in the problems of soliciting customer preferences regarding a set of products, establishing funding priorities of a set of projects, establishing searching priorities for a set of search engines in the Internet. In this thesis, we have improved previously developed MST (Majority Spanning Tree) algorithm for solving this problem, where the number of violations has been chosen as the criterion of optimality. Whereas in previous MST algorithms a subset  $A$  of consecutively ranked players could be swapped with a subset  $B$  of consecutively ranked players, where  $A$  and  $B$  are consecutively ranked, in this improved version we have allowed  $A$  and  $B$  to be any disjoint subset. We also developed another Hybrid algorithm using five related algorithms (Sort, Hamiltonian path, Arrange, MST and Improved MST) and compare the performance of these algorithms for the same sets of input data. Experimental results suggest that these new algorithm outperforms the existing ones.

# Contents

Board of Examiners	ii
Dedication	iv
Candidate's Declaration	v
Acknowledgment	vi
Abstract	vii
1 Introduction	1
1.1 The Round Robin Tournament Problem .....	3
1.2 Literature Survey .....	4
1.2.1 Time Relaxed Single Round Robin Tournament Problem (TRSRR)	7
1.2.2 Time Relaxed Double Round Robin Tournament (TRDRRT).....	7
1.3 Objective of the Thesis .....	8
1.4 Organization of the Thesis .....	8
2 Existing Algorithms	9
2.1 Iterated Kendall Algorithm.....	9
2.2 Generalized Iterated Kendall Algorithm.....	11

2.3	Hamiltonian Path Algorithm.....	13
2.4	Arrange Algorithm .....	15
2.5	The Feedback Arc Set Problem .....	16
2.5.1	Some Applications of FAS.....	17
2.6	Sorting Algorithms.....	18
2.7	A PTAS for Weighted Feedback Arc Set on Tournaments .....	19
3	Majority Spanning Tree	20
3.1	Introduction to MST .....	20
3.2	Existing MST Algorithm For Round Robin Tournament Problem .....	22
3.2.1	Majority Spanning Tree Algorithm .....	22
3.2.2	A Modified Algorithm For Round-Robin Tournament .....	25
3.3	Limitations in Existing Algorithms.....	28
3.4	Improved MST Algorithm .....	28
3.4.1	Procedure of Improved MST.....	30
3.5	Hybrid Algorithm.....	32
4	Implementations and Experiments	35
4.1	Implementation .....	35
4.1.1	Implementation of Sort .....	35
4.1.2	Implementation of HP.....	36
4.1.3	Implementation of Arrange .....	36
4.1.4	Implementation of MST .....	36
4.1.5	Implementation of Improved MST .....	37
4.2	Experiments.....	37

4.2.1	Graphical Analysis of the Results .....	44
5	Conclusion	47
5.1	Contribution .....	47
5.2	Future Works .....	48
References		

# List of Figures

1.1	Representation of results of a Round-Robin Tournament in a digraph . . .	4
2.1	Procedure of Iterated Kendall algorithm .....	10
2.2	Before applying Hamiltonian Path algorithm .....	13
2.3	After applying Hamiltonian Path algorithm .....	13
3.1	Example of fundamental cutset.....	21
3.2	Majority Spanning Tree.....	22
3.3	A Tournament digraph and a matrix of the result of the tournament.....	30
3.4	The algorithm divides the graph into three sets.....	30
3.5	Situation of digraph after swapping set I and set K.....	31
3.6	Situation of digraph after swapping set I and set J.....	32
3.7	Situation of digraph after swapping set J and set K .....	32
4.1	Comparison among MST, Improved MST and Hybrid Algorithm.....	44
4.2	Situation in worst case .....	45
4.3	Situation in best case .....	45
4.4	Average computational time required by each algorithm.....	46



# List of Tables

1.1	Four team Round Robin Tournament.....	2
1.2	Five team Round Robin Tournament .....	2
1.3	Representation of the game result.....	4
1.4	Time Relaxed Single Round Robin Tournament .....	7
4.1	Number of upsets for different algorithms .....	38
4.2	Average number of upsets for different algorithms .....	42
4.3	Average computational time of three algorithms in seconds .....	43
4.4	Ranking of various algorithms for 10 players.....	44

# Chapter 1

## Introduction

A round-robin tournament (or all-play-all tournament) is a competition in which each contestant plays all other contestants in turn. The term round-robin is derived from the French term *ruban*, meaning “ribbon”. Over a long period of time, the term was corrupted and idiomized to *robin*. In the United Kingdom, a round-robin tournament is often called an American tournament in sports such as tennis or billiard which are usually knockout tournaments. In Italian it is called *girone all’italiana* (literally “Italian-style circuit”).

In theory, a round-robin tournament is the fairest way to determine the champion among a known and fixed number of participants. Each participant, player or team, has equal chances against all other opposites.

Round Robin scheduling is interesting in its own right. Some leagues have a schedule that is a single or double round-robin schedule. Examples of this include many U.S college basketball leagues and many European football leagues. For such leagues, the scheduling problem is exactly a constrained round-robin scheduling problem, where the constraints are generated by team requirements, league rules, media needs, and so on [27].

If  $n$  is the number of competitors, a pure round robin tournament requires  $\binom{n}{2}$  games. If  $n$  is even, then in each of  $\frac{n}{2}$  rounds,  $\frac{n}{2}$  games can be run concurrently. If  $n$  is odd, there will be  $\lceil \frac{n}{2} \rceil$  rounds, each with  $\lfloor \frac{n}{2} \rfloor$  games, and one competitor having no game in that round.

The table 1.1 is an example of four team round-robin tournament.

Team	Win	Loss
1		
2		
3		
4		

Round1	Round2	Round3
1 vs 2	1 vs 4	1 vs 3
4 vs 3	3 vs 2	2 vs 4

Table 1.1: Four team Round Robin Tournament

The table 1.2 is an example of five team round-robin tournament.

Team	Win	Loss
1		
2		
3		
4		
5		

Round1	Round2	Round3	Round4	Round5
1 vs 4	3 vs 1	5 vs 3	2 vs 5	4 vs 2
2 vs 3	4 vs 5	1 vs 2	3 vs 4	5 vs 1
5-bye	2-bye	4-bye	1-bye	3-bye

Table 1.2: Five team Round Robin Tournament

## 1.1 The Round Robin Tournament Problem

Ranking is a fundamental activity for organizing and, later, understanding data. Advice of the form “a should be ranked before b” may be given by ranking [8, 26, 25]. If this advice is consistent, and complete, then there is a total ordering on the data and the ranking problem is essentially a sorting problem. If the advice is consistent, but incomplete, then the problem becomes topological sorting. If the advice is inconsistent, then we have the Feedback Arc Set (FAS) problem: the aim is then to rank a set of items to satisfy as much of the advice as possible. An instance in which there is advice about every pair of items is known as a tournament. This ranking task is equivalent to ordering the nodes of a given directed graph from left to right, whilst minimising the number of arcs pointing left [8].

The problem of ranking players in a round-robin tournament, in which outcome of any match is a win or a loss, is to rank players according to their performances in the tournament.

It is known that the results of a tournament can be represented in a digraph,  $G = (V, A)$  known as tournament graph, where vertices correspond to players and arcs correspond to match results. A tournament result is said to be upset (or violation) if a lowly-ranked player has defeated a highly-ranked player [11]. Our goal is to reduce the number of upsets as much as possible.

The problem of minimizing the number of upsets is equivalent to finding the minimum number of arcs in a digraph deletion of which results in an acyclic digraph. This problem is known as the Minimum Feedback Arcset Problem, which is NP-hard [13, 15, 25]. A classical result of Lawler and Karp [18] asserts that finding a minimum feedback arc set in a digraph is NP-hard. The minimum FAS for tournaments is polynomially equivalent to the minimum FAS for digraphs, and thus also NP-hard.

Theorem 1.1.1. The minimum feedback arc set for tournaments is NP-hard [7].

Let us consider a simple scenario where five players (a, b, c, d, e) have participated in a Round Robin Tournament. Figure 1.1 shows the graphical representation of the game. From figure if we look at player A, we see there are three incoming arcs and one outgoing arc which means A has lost three times and has won just one time. Player A is defeated by player B, C and D won against E. Similarly, player B is defeated by C and E and won against A and D and so on.

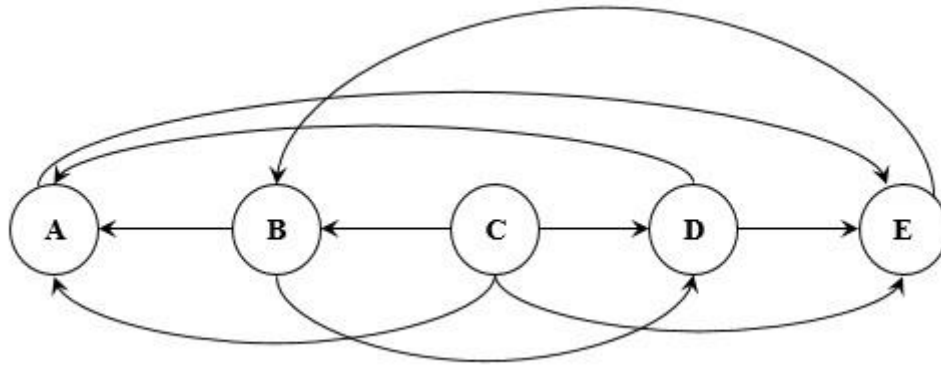


Figure 1.1: Representation of results of a Round-Robin Tournament in a digraph

	E	D	C	B	A
A	1	0	0	0	
B	0	1	0		
C	1	0			
D	0				
E					

Table 1.3: Representation of the game result

Table 1.3 represents the result of the tournament with 0 indicating a loss and 1 indicating a win. For understanding result we have to traverse each row from left to right. Since A has won against E we have 1 in the cell. Let us analyze the situation of player A in respect to other players. First consider A with E. Here 1 represents A has won the match with E as A is in the row and E is in the column. Then in case of A and D, 0 means A has lost the match. Similarly 0 and 1 represents the match status for other players.

## 1.2 Literature Survey

The Round Robin tournament structure also arises in other environments, for example, establishing searching priorities for a set of search engines in the internet. In [28], Ka Wai and Chi Ho present an algorithm for merging results from different data sources in meta-search engine. They further extend one that has developed for ranking players of a round-robin

tournament to a more general one when the ranking input is given from multiple sources. The problem in meta-search engine can be represented by a complete directed graph which can be used by the Majority Spanning Tree (MST) algorithm [19].

In the situation where a consumer or respondent in a market survey specifies preferences pertaining to a set of products, these preferences are often given in the form of pairwise comparisons (product  $i$  is preferred to product  $j$ ). In the case where such a comparison is made between all pairs then a binary matrix of the tournament type would result. The ranking of this tournament would constitute a preference ordering of the products for this particular respondent. Techniques such as those discussed in Cook and Seiford [10], e.g., could then be utilized to derive a group preference or consensus of opinions. The tournament approach, therefore, becomes a valuable technique for processing consumer preferences involving pairwise comparisons.

In any organization regular assessments of personnel are done as a matter of course. It is particularly true in the case of the military, e.g., that officers as well as enlisted personnel undergo performance appraisal on an annual basis. While it is generally true that each individual is to be assessed on his/her own merits, it is also the case that relative comparisons are often made, particularly when promotion quotas are enforced. In the final analysis some form of pairwise comparison of candidates would or could be invoked to arrive at a ranking of these candidates. In this instance the same tournament structure could be created, with the modification that some pairs may not be compared.

The problem of establishing funding priorities for a set of projects (e.g. in a transportation department) can also give rise to the tournament model. This is particularly true in a multicriteria project evaluation situation. Here a common approach is to apply a non-compensatory method such as concordance analysis [23]. Briefly, the concordance model is constructed as follows: Assign a rating  $r_{ij}$  to each project  $i$  relative to criterion  $j$ . Letting  $w_j$  denote the weight or importance to be attached to criterion  $j$ , determine for each pair  $(i, i')$  of projects a concordance index  $s_{ii'}$  and a discordance index  $d_{ii'}$ , where

$$S_{ii'} = \frac{\text{Sum of the weights of those criteria where project } i \text{ is rated equal to or better than } i'}{\text{Sum of all weights}}$$

$d_{ij}$  = Largest negative difference (i.e. i rated higher than j) between ratings for a single criterion  
 The difference between the maximum obtainable and minimum obtainable ratings.

Using thresholds  $T_s$  and  $T_d$ , compute a preference index

$$a_{ij} = \begin{cases} 1, & \text{if } S_{ij} \geq T_s \text{ and } d_{ij} \leq T_d \\ 0, & \text{otherwise.} \end{cases}$$

The matrix  $A = (a_{ij})$  is then used to rank the projects. These and related applications are discussed in [1]

In [27], Michael A. Trick has worked on a large practical scheduling problem, that is Major League Baseball (MLB). Fully defining the MLB schedule is a daunting task, requiring the collection of more than 100 pages of team requirements and requests, along with an extensive set of league practices. The key inside into effectively scheduling MLB, however, was the recognition that the complicated schedule could generally be broken into various phases, where each phase consists of a round-robin schedule, sometimes among subsets of teams.

Round robin tournament format is also used in National Basketball Association (NBA). Bao [5, 16] examine the properties of general time-relaxed round robin tournaments and also defined single round, double round and multiple round time-relaxed round robin tournament.

In modern days, most leagues use a round robin tournament schedule format which means each team play every other team for a fixed number of times during a time span, which is called a round. The round robin tournament schedules can be divided into two broad types: time constrained schedules and time relaxed schedules. In time constrained schedules, the number of available game slots is equal to necessary game slots. The time constrained schedules are used by many leagues, including most college basketball conferences, professional soccer leagues in the Europe and the South America. In the time relaxed schedules, the time of available game slots is bigger than the necessary number. It is used by a few leagues, the National Basketball Association (NBA) and the National Hockey League (NHL) in North America are two examples. The NBA is the most popular professional basketball league in North America. Bao [5] was inspired by the scheduling problem for the NBA regular season.

The round-robin is widely used in many other leagues, especially amateur sports leagues. As a result of the limited arena and player availability, round robin tournament format is the only choice for most amateur leagues.

### 1.2.1 Time Relaxed Single Round Robin Tournament Problem (TRSRR)

Time relaxed single round robin tournaments have the general round robin tournament structure requirement: every team will meet every other team at a fixed number, which is one in this case. In a time constrained round robin tournament, the number of the available game days is equal to the minimum required days. Unlike its counterpart, time relaxed round robin tournaments have more game days than the minimum required. Therefore it is necessary to define the parameter of the number of available game days. Let, the number of available time slots is double size of the number of the games. For instance, there are  $2(n - 1)$  time slots available if a team has to play  $n - 1$  games. Table 1.4 is an example schedule for a tournament with six teams.

Day	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10
Team 1	vs 3		vs 2	vs 6	vs 5		vs 4			
Team 2			vs 1	vs 4		vs 5	vs 3			vs 6
Team 3	vs 1	vs 6		vs 4		vs 2	vs 5			
Team 4	vs 6	vs 5		vs 2	vs 3		vs 1			
Team 5		vs 4			vs 1	vs 2		vs 3	vs 6	
Team 6	vs 4	vs 3		vs 1					vs 5	vs 2

Table 1.4: Time Relaxed Single Round Robin Tournament

### 1.2.2 Time Relaxed Double Round Robin Tournament (TRDRRT)

Teams play each other twice in a double round robin tournament, normally one at home, the other on road. In the time constrained schedules, the mirrored double round robin tournament is popular. A mirrored double round robin tournament consists of two rounds with identical timetables, and the venues in the second round are reversed to those in the first round. For example, if team  $i$  play at home with team  $j$  on day  $d$  ( $d \leq \frac{D}{2}$ ), then team  $j$  will



play at home with team  $i$  on day  $d + 2D$ .

### 1.3 Objective of the Thesis

In this thesis, we concentrate to give a better ranking for the players or teams of a round-robin tournament, so that the number of violations reduces as much as possible. Although a number of techniques have been developed to solve the problem of Ranking player in Round-Robin tournament. We improve these algorithms further to obtain better solutions. Therefore, the main objectives of our thesis are as follows:

- First is to examine the properties of the round robin tournament problem structure and to analyze and understand various Ranking algorithms.
- Second is to develop a new heuristic algorithm, which will give better results compared to the previous algorithms.
- Third is extend this algorithm for solution of related problems.

### 1.4 Organization of the Thesis

The work is organized as follows. Chapter 1 introduces the round-robin tournament structure and also explains the round-robin tournament problem in details. An extensive survey about existing algorithms for the round robin tournaments is given in Chapter 2. A detailed description about Majority Spanning Tree (MST), previously developed MST algorithms for round-robin tournaments and our new inventions Improved MST algorithm and Hybrid algorithm are depicts in Chapter 3. Chapter 4 is divided into two main parts: part-1 illustrates the implementation process of Sort, HP, Arrange, MST and Improved MST, experimental results in two different forms (table and graph) is described in part-2. Comparison of up-set and computational time in these five algorithms is also presented in this Chapter. Our contribution on this thesis and some suggestion for future work is given in Chapter 5.

# Chapter 2

## Existing Algorithms

In this Chapter, we provide an overview of round-robin tournament ranking algorithms that focus on reducing number of violations among the players or teams. A significant amount of works have been done to alleviate the round-robin tournament ranking problem. One of the earliest proposed was that based on Kendall scores [20, 2]. In this method, players are ranked according to the number of opponents each defeats. Section 2.1 discusses Iterated Kendall(IK) algorithm. An Improved version of IK is Generalized Iterated Kendall, discussed in Section 2.2. Section 2.3 describes how Hamiltonian Path (HP) algorithm works and Section 2.4 reviews the working procedure of Arrange algorithm. We have already said that this problem is similar to Minimum Feedback Arcset Problem(FAS), so we have also studied some algorithms which are related to FAS problem. In Section 2.5 we illustrate the Feedback Arcset Problem. A Polynomial Time Approximation Scheme (PTAS) to solve FAS problem on tournaments is discussed in Section 2.7.

### 2.1 Iterated Kendall Algorithm

In Ali et al. [1] the IK algorithm is presented. This algorithm is an extension of the Kendall scores approach, and is guaranteed to produce a ranking with no ties. The method works as follows:

1. Compute the Kendall scores by finding the row sums of the tournament matrix  $A(T)$ , and rank order the players according to these scores. If there are no ties, the resulting

ranking will produce no violations and the procedure terminates.

2. If  $k$  players are tied for some position, break the ties by considering the submatrix of  $A(T)$  containing only the rows and columns of these  $k$  players, and performing the ranking as in step (1).
3. If there are  $l$  players who are tied among themselves (i.e. the row sums of the  $1 \times 1$  submatrix for these players are all equal), select any one of the  $l$  players and place it first in the subranking. The tie among the remaining  $l - 1$  players is broken by performing step (2).

Referring to the previous example in which players (c, d, e) tied in the row sums, step (2) of the IK algorithm indicates that row sums for the submatrix for these 3 players should be computed. Specifically,

	a	b	c	d	e	row sums
a						
b						
c			0	0	1	1
d			1	0	0	1
e			0	1	0	1

Figure 2.1: Procedure of Iterated Kendall algorithm

Since there is still a tie in these row sums, we randomly select one of the players (c, d, e) to be the first in the submatrix—say player c (step3). Returning to step (2) with the remaining submatrix corresponding to de, we get row sums

$$d \rightarrow 0$$

$$e \rightarrow 1$$

According to step (2) of the algorithm, player e should be ranked next after c. The final ranking is then  $a > c > e > d > b$ .

## 2.2 Generalized Iterated Kendall Algorithm

In the GIK algorithm restoring is done by using that sub-tournament involving all players not yet ranked. In case there is still a tie in this sub-tournament, the GIK procedure tries to find a player who defeated the last player put in the ranking. If such a player exists, it is advantageous to put that player next, and then immediately change his place with the previously last ranked player. In this manner, a reduction in the overall number of violations by one (as compared with a random positioning) is guaranteed.

The GIK algorithm appears below. The following conventions are used.

- $|S_1|$  denotes the cardinality of the set of players.
- $\emptyset$  denotes the empty set.
- If  $S_1$ , and  $S_2$  denote sets (subsets) of players, then  $S_1/S_2$ , denotes those players in  $S_1$ , but not in  $S_2$ .
- If  $R$  denotes a ranking and  $P$  a player,  $R||P$  denotes the ranking formed by placing player  $P$  after the last player in the ranking  $R$ .
- Given  $R_1 = (P_1 > P_2 > \dots > P_k)$  and  $R_2 = (Q_1 > Q_2 > \dots > Q_l)$ , then  $R_1||R_2 = (P_1 > P_2 \dots > P_k > Q_1 > Q_2 > \dots > Q_l)$ .

In the following algorithm,  $A$  will denote the set of unranked players. Initially, the players will all be scored according to the row sums as in the IK algorithm. Let the set  $D$  include all players that have the maximum score. There might be one such player or several. This set  $D$  will be referred to as the dominant set in  $A$ . As set  $A$  changes, so also will set  $D$ .

---

### Algorithm 1 GIK Algorithm

---

- 1: Let  $R = \emptyset$ ,  $A = \{P_1, P_2, \dots, P_n\}$ .
  - 2: if  $A = \emptyset$  then go to (15); otherwise determine the current scores of players in  $A$ .
  - 3: if  $A = \emptyset$  then go to (15); otherwise determine  $D$ , the dominant set.
  - 4: If  $|D| > 1$ , then go to (6).
  - 5: Letting  $P$  denote the only player in  $D$ , form the ranking  $R = R||P$ , let  $A = A \setminus \{P\}$  and go to (3).
-

---

Algorithm 1 GIK Algorithm Continued...

---

- 6: If from the last time of updating the current scores of A [step (2)], set A has changed, then go to (2).
  - 7: If  $D > 2$ , then go to (9).
  - 8: Let  $P_1$  and  $P_2$ , denote the players in D with  $P_1 > P_2$ . Let  $R = R||P_1||P_2$ , and  $A = A\{P_1, P_2\}$ . Go to (2).
  - 9: If  $R = 0$ , then go to (11).
  - 10: Arrange all players in D in Hamiltonian order H, i.e.  $H = (P_1 > P_2 > \dots > P_k)$ . Let  $R = R||H$  and  $A = A\{P_1, P_2, \dots, P_k\}$ . Go to (2).
  - 11: Let Q denote the last player presently in R, and let  $\{P_1, P_2, \dots, P_k\}$  constitute D. Let  $i = 1$ .
  - 12: If  $i > k$ , go to(10).
  - 13: If  $P_i > Q$ , put  $P_i$  in R ahead of Q. Let  $A = A\{P_i\}$  and  $D = D/\{P_i\}$ . If  $|D| = 0$ , then go to (2). Otherwise go to (4).
  - 14: Let  $i = i + 1$ , and go to (12).
  - 15: Execute procedure Arrange on the ranking R.
  - 16: End.
- 

Once the first set of scores for all players is determined (step 2), steps 3 and 5 are carried out to determine the initial partial ordering among the dominant set of untied players. As soon as the first block of ties is reached ( $|D| > 1$ ), an attempt is made to rank this block in a manner which is advantageous. If  $|D| = 2$ , then step 8 arranges the two dominant players in the remaining set A according to their tournament outcome. If  $|D| > 2$ , then steps 11, 12, 13 and 14 attempt to find a player in D who defeated the last ranked player Q in R. If one is found, then this player is ranked ahead of Q and we continue to look for the next player (among those remaining ones in D) who defeated Q etc. When all players have been ranked, procedure Arrange, described earlier, is executed (step 15).

Theorem 2.2.1. The complexity of the GIK algorithm is  $O(n^4)$  [9].

Proof. Step 2 requires  $O(n^2)$ , step 3,  $O(n)$  and step 10,  $O(k^2)$  operations. Loop 12-13-14 can be executed at most k times, and from step 13 we can proceed to step 4 at most  $k - 2$  times. We can return to step 2 only if A is changed, which can occur at most  $n - 1$  times.

Taking all this into consideration, after  $O(n^3)$  operations, step 15 is reached, which requires at most  $O(n^4)$  operations as discussed above. Thus the algorithm is  $O(n^4)$ .  $\square$

## 2.3 Hamiltonian Path Algorithm

In the mathematical field of graph theory, a Hamiltonian path is a path in an undirected or directed graph that visits each vertex exactly once. A Hamiltonian cycle (or Hamiltonian circuit) is a Hamiltonian path that is a cycle.

The Hamiltonian path algorithm finds a ranking of players such that a player always defeats a player ranked immediately below him. For example consider the following graph of five players A, B, C, D and E.

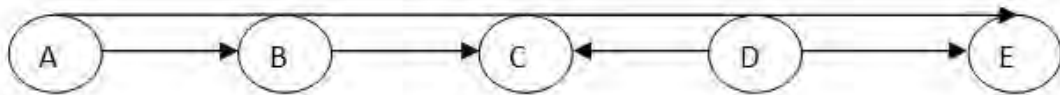


Figure 2.2: Before applying Hamiltonian Path algorithm



Figure 2.3: After applying Hamiltonian Path algorithm

From Figure 2.3 we observed that A defeats B, B defeats D, D defeats E and E defeats C. So the ranking is  $A > B > D > E > C$ .

**Definition 2.3.1.** A ranking  $R$  of a tournament is said to be Hamiltonian if for any  $i$  ( $i = 1, 2, \dots, n$ ), the player ranked in  $i^{\text{th}}$  position has defeated (in the tournament) the player ranked in position  $(i + 1)$  [9].

Such a ranking is referred to as a Hamiltonian Path. From the above example it is clear that significant improvements, in terms of reducing the number of violations, may be obtainable when a nonHamiltonian ranking is transformed into a Hamiltonian ranking. The following algorithm can be utilized to convert any given ranking into a Hamiltonian Path.

---

**Algorithm 2 Hamiltonian Path Algorithm 1:**

---

Let  $i = 1$ .

2: Let  $l = i$ .

3: If  $P_i > P_{i+1}$ , then go to (9).

4: Interchange  $P_i$  and  $P_{i+1}$  (i.e. let  $P_i = P_{i+1}$  and  $P_{i+1} = P_i$ ). 5:

Let  $K = 1$ . If  $K = 1$ , then go to (9).

6: If  $P_{K-1} > P_K$ , then go to (9).

7: Interchange  $P_{K-1}$  and  $P_K$  and set  $K = K - 1$ . 8:

If  $K > 1$ , then go to (6).

9: Set  $i = i + 1$ .

10: If  $i < n$ , then go to (2).

11: End.

---

In step 3, the algorithm determines if the  $i^{\text{th}}$  ranked player actually defeated the  $(i + 1)^{\text{st}}$  ranked player. If so then they are sequenced in the “proper” order. We then update  $i$  (step 9), and check the next consecutive pair in the sequence. If player  $i$  did not defeat player  $i + 1$  in the tournament, then these two players are interchanged in the ranking (step 4).

Each time a pair of players is interchanged, it is necessary to back up through the ranking just examined to see if the new player  $P_i$  (which was player  $P_{i+1}$ ) must be interchanged with player  $P_{i-1}$ , etc. This is accomplished in steps 5 to 8. Since this is not the case, the ranking is Hamiltonian. Since the outer loop (steps 2 to 10) is repeated  $n$  times, and since the inner loop (steps 6 to 8) can involve at most  $n$  passes, the following theorem holds.

**Theorem 2.3.1.** The Hamiltonian Path algorithm converges in at most  $O(n^2)$  operations. The Hamiltonian Path algorithm checks only to determine if two adjacent players can be switched [9].

Limitations in Hamiltonian Path algorithm:

Only consider a ranking of the players such that each player lost to the player ranked one position higher. Do not observe the total scenario.

## 2.4 Arrange Algorithm

ARRANGE algorithm starts with an arbitrary ranking and improves it by rearranging the ranking of a single player so that the total number of violations is decreased. Starting with a given ranking  $R = (P_1, P_2, \dots, P_n)$  of  $n$  players in a tournament, an improvement in the ranking (reduction in the number of violations) can be attempted by determining whether or not each player is ranked in the “best” possible position. Specifically, a check can be made to determine whether or not the moving of a player to some new position in the ranking, while keeping all other players fixed in their respective positions, will reduce the number of violations. If so, a new ranking  $R'$  is created as follows.

Starting with  $R$ , consider moving some player  $P_i$  to a new position  $k$  ( $k \neq i$ ). This gives rise to the new ranking

$$R' = (P_1, P_2, \dots, P_{k-1}, P_i, P_k, \dots, P_{i-1}, P_{i+1} \dots P_n)$$

Now, generally this move would be executed if and only if the number of violations in  $R'$  will be strictly less than that of  $R$ , with one exception. The move to create  $R'$  with  $R'$  having the same number of violations as  $R$  will be carried out, if and only if player  $P_{i+1}$  is defeated by player  $P_{i-1}$ . In this instance  $R'$  is easily converted to  $R''$ , in which  $P_{i-1}$  and  $P_{i+1}$  are interchanged, i.e.

$$R'' = (P_1, P_2, \dots, P_{k-1}, P_i, P_k, \dots, P_{i+1}, P_{i-1} \dots P_n)$$

$R''$  has 1 less violation than did  $R$ . In order to determine the number of violations that will result by moving  $P_i$  to a new position  $k$ , only  $O(n)$  operations rather than  $O(n^2)$  are required. The moving of  $P_i$  to position  $k$  can be regarded as  $k - i$  movements (from  $i$  to  $i + 1$ , from  $i + 1$  to  $i + 2$ , and so on). The number of violations for each new ranking (formed by moving  $P_i$  from position  $j$  to  $j + 1$  say) differs from the number for the previous ranking by  $+1$  or  $-1$ , and thus can be computed by  $O(1)$  steps. Therefore, the number of violations of the new ranking can be computed in  $O(k - i)$  operations. Since  $k - i < n$  then the number of operations is  $O(n)$ .

The instructions for procedure Arrange appear below. Steps 3 and 6 require  $O(n)$  operations



each, while all other steps require  $O(1)$  operations. The main loop from step 2 to step 9 is done  $n$  times. In case no new ranking is formed, the algorithm requires  $O(n^2)$  steps. In case new rankings are formed, reducing the number of violations from  $V_1$  to  $V_2$ , then because after each new ranking is formed we return to step 1, the number of steps is  $(V_1 - V_2) \times O(n^2)$ . In the worst possible case, the number of violations is  $O(n^2)$ . Hence, at most  $O(n^2)$  new rankings are formed, and the maximum number of steps is  $O(n^4)$ .

Theorem 2.4.1. The Arrange procedure converges in at most  $O(n^4)$  operations [9].

---

### Algorithm 3 Arrange Algorithm

---

- 1: Let  $i = 1$ , and go to (3).
  - 2: if  $P_{i+1}$  defeats  $P_{i-1}$ , then go to (6).
  - 3: For  $j = 1$  to  $n$ , move  $P_i$  to position  $j$  if and only if a better ranking is achieved. 4: If a better ranking is obtained, go to (1).
  - 5: Go to (8).
  - 6: For  $j = 1$  to  $n$  ( $j = i$ ), move  $P_i$  to position  $j$  if and only if a ranking is obtained which has no more violations than is true of the present one.
  - 7: If  $P_i$  is moved, interchange  $P_{i+1}$  and  $P_{i-1}$  to get a better ranking, and go to step (1). 8: Set  $i = i + 1$ .
  - 9: If  $i < n$ , then go to (2).
  - 10: End.
- 

Limitations in Arrange algorithm:

Only works upon single player but not for a set of players.

## 2.5 The Feedback Arc Set Problem

The Feedback Arc Set (FAS) problem is a key combinatorial problem: to rank items in a set given only advice about the correct way to order specific pairs. A ranking of a set is simply a permutation of that set. Thus the only information we have to help us to form our ranking is a set of statements of the form 'a should be ranked before b'. If each statement is consistent with all others, the problem is simple to solve— just return a ranking that agrees with all the advice. However, difficulties arise when there is contradictory, or inconsistent,

advice. For instance, given the three statements, ‘a should be ranked before b’, ‘b should be ranked before c’ and ‘c should be ranked before a’, there is no ranking of a, b and c which agrees with all the statements. The difficulty of the FAS problem, much like problems of clustering with advice— such as the correlation clustering problem [4] — is in deciding which of the inconsistent advice to follow, and which to violate. The aim is to minimize the number of statements that are violated. The natural graph representation of the problem uses a vertex for each item and a directed arc from a to b for each demand ‘a should be ranked before b’. In this context, the aim is to order the vertices from left to right so that the number of arcs pointing left (back-arcs) is as small as possible.

Problem: Feedback Arc Set (FAS)

Given a directed graph  $G = (V, E)$ , find an ordering over  $V$  to minimize the number of back arcs: that is the number of arcs

$$\{v \rightarrow w \in E \mid \pi(v) > \pi(w)\}$$

Given some ranking  $\pi$ , if the set of back-arcs is removed, then all cycles in the graph will eliminate. Such a set is called a Feedback Arc Set. An equivalent formulation of the problem is therefore: given a digraph  $G$ , find the smallest subset  $S$  of the arcs of  $G$  that intersects all cycles in  $G$ . The graph  $G'$  obtained by removing the arcs in  $S$  from  $G$  is acyclic and thus a DAG—and admits a consistent ordering via a topological sort.

### 2.5.1 Some Applications of FAS

Originally motivated by problems in circuit design [17], FAS has found applications in many areas, including computational chemistry [22, 24], and graph drawing [14]. Closely related to the FAS problem is the Rank Aggregation problem.

Problem: Rank Aggregation

Given a set  $\Pi$  of rankings over a set  $V$ , find a ranking  $\sigma$  to minimise  $\sum_{\pi \in \Pi} K(\sigma, \pi)$ , where  $K(\pi, \sigma)$  is the Kemeny distance [1959]; defined to be the number of pairs  $v, w \in V$  where  $\pi(v) < \pi(w)$  and  $\sigma(v) > \sigma(w)$ .

Dwork et al. [12] outline the problem and motivate it as a method for aggregating data from

search engines. There is a significant body of work studying this problem, which is known as MetaSearch [3].

Rank Aggregation is a special case of FAS on weighted tournaments. There is a fairly simple reduction: for each  $v, w \in V$ , if  $v$  is ranked above  $w$  in some fraction  $f$  of the input rankings, place an arc between  $v$  and  $w$  with weight  $f$ . The connection to Rank Aggregation is a principal reason for the attention paid to the FAS problem on tournaments.

## 2.6 Sorting Algorithms

It is possible to define a FAS algorithm which is analogous to almost any sorting algorithm. Unlike traditional sorting problems, in which we assume there is a total order on the data, the difficulty in FAS is the lack of transitivity, which sorting algorithms are designed to exploit. Nevertheless, sorting algorithms provide schemes for deciding which of the advice to believe. So we can define a general strategy for FAS based on some sorting algorithm  $S$ ; run  $S$  over the vertices of  $G$ , using as the comparison function “ $u < v$  if and only if  $u \rightarrow v$ ”. For instance, using this strategy, Quicksort is defined as follows:

Algorithm: Quicksort

Choose a pivot  $p \in V$ , uniformly at random. Let  $L \subseteq V$  be all vertices  $v$  such that  $v \in p$ , and let  $R = V \setminus (L \cup \{p\})$ . Also, let  $\pi_L$  be the ordering of  $L$  obtained by Quicksort, and  $\pi_R$  be the analogous ordering of  $R$ . Output  $(\pi_L, v, \pi_R)$ , the ordering resulting from placing vertices in  $L$  on the left (ordered by  $\pi_L$ ), etc.

Cook et al. [9] focus on ensuring a Hamiltonian path exists along the final ordering of the nodes; any sensible algorithm should achieve this. The method they use to achieve this is in effect a bubble sort of the tournament. Chanas and Kobylanski [6] apply an insertion technique to the Linear Ordering problem that is more involved than the usual Insertionsort. As a subroutine, they use what is in effect Insertionsort, which they name SORT. It is defined as follows:

Algorithm: Sort

Make a single pass through the nodes from the left to the right. As each node is considered, it is moved to the position to the left of its current position that minimises the number of

back-arcs (if that number of back arcs is fewer than its current position).

## 2.7 A PTAS for Weighted Feedback Arc Set on Tournaments

In [21] Claire et al. gives a polynomial time approximation scheme for solving FAS on tournaments. They use existing constant factor approximation algorithms as well as polynomial-time approximation schemes for the complementary maximization problem.

**Theorem 2.7.1. (PTAS).** There is a randomized polynomial-time approximation scheme for minimum weighted Feedback Arc Set on tournaments and for Kemeny-Young rank aggregation. Given  $\epsilon > 0$ , the algorithm outputs, in time  $O((1/\epsilon)n^6 + 2^{2O(b/\epsilon)}n^4)$ , an ordering whose expected cost is less than  $(1 + \epsilon)$  OPT. The algorithm can be derandomized at the cost of increasing the running time by a factor of  $n^{2O(1/\epsilon)}$  [21].

---

### Algorithm 4 Polynomial Time Approximation Scheme of Theorem 2.7.1

---

Given: Fixed parameters  $\epsilon > 0$  and  $b \in (0, 1]$ .

Input: A weighted tournament.

Round each weight to the nearest integer multiple of  $\epsilon b/n^2$ .

$\pi \leftarrow$  output of any constant factor approximation algorithm.

While there exists a cost-decreasing move, do that move. The two types of moves are:

1. Single vertex moves. Choose a vertex  $x$  and a rank  $j$ , take  $x$  out of the ordering  $\pi$  and insert it back in so that its rank is  $j$ .
2. Additive approximation. Choose two integers  $i < j$ ; let  $U$  be the set of vertices whose current ranks are in  $[i, j]$ . Execute the derandomized version of algorithm AddApprox on  $U$ , with  $\beta = 9^{-r_{\max}} \epsilon^3$ . Let  $\pi' \cup d$  be the result. Replace the restriction  $\pi|_U$  of  $\pi$  to  $U$  by  $\pi' \cup d$ .

Output:  $\pi$

---

# Chapter 3

## Majority Spanning Tree

In this Chapter, we provide an overview of Majority Spanning Tree (MST) and MST algorithms that focus on reducing violations of round-robin tournament. Section 3.1 discusses definition and properties of Majority Spanning Tree (MST). Two MST algorithms which are already developed for solving the round-robin tournament problem are reviewed in Section 3.2. Section 3.3 identifies some problems of existing algorithms. Finally, Section 3.4 shows the new modification of MST algorithm and Section 3.5 discussed our another invention Hybrid algorithm.

### 3.1 Introduction to MST

For understanding majority spanning tree (MST) first we need to know what is fundamental cutset. Fundamental cutset is defined by-

$(V - 1) * (V - 1)$  matrix  $Q = (q_{ij})$

$$q_{kj} = \begin{cases} 1, & \text{if edge } j \text{ is in the positive orientation of cutset } k \\ -1, & \text{if edge } j \text{ is in the negative orientation of cutset } k \\ 0, & \text{Otherwise} \end{cases}$$

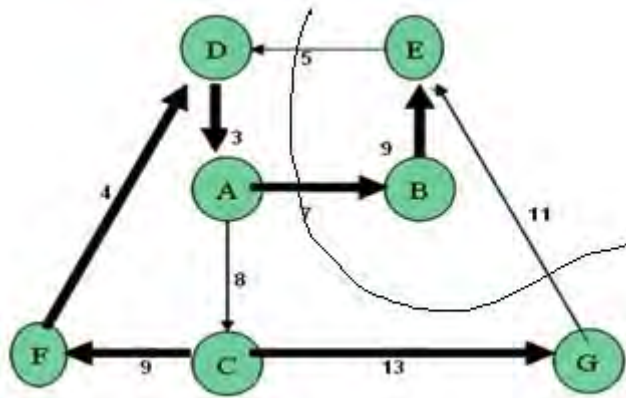


Figure 3.1: Example of fundamental cutset

Let  $T$  be any arbitrary spanning tree of  $G$ ,  $K_k$  be the fundamental cutset defined by edge  $(i_1, j_1) = e_k \in T$ , and

$$r_k = \sum_{(i,j) \in K_k^+} P_{ij} - \sum_{(i,j) \in K_k^-} P_{ij}$$

where,  $K_k^+$  is the set of forward edges of the cutset  $K_k$ , and  $K_k^-$  is the set of reverse edges of the cutset  $K_k$ . We define the value of the cutset  $K_k$  by  $P(K_k) = r_k$ ,  $P_{ij}$  is the weight of the edge  $(i, j)$ .

If  $e_k = (AB)$  then  $K_k^+ = \{AB, GE\}$ ,  $K_k^- = \{ED\}$  So,  $r_k = 7 + 11 - 5 = 13$

Definition of MST: A spanning tree  $T$  is said to be a majority spanning tree of the digraph  $G = (V, E)$  with real weight function  $P : E \rightarrow R^+$  if for each fundamental cutset  $K_k$ , determined by the edges of  $T$ ,  $P(K_k) \geq 0$

For example consider the figure 3.2, the value of the cutset determined by the edge  $DA$  is  $(-5 + 3 - 8 + 11) = 1 \geq 0$  since  $DA$  and  $GE$  are in the same orientation whereas  $AC$  and  $ED$  are in the opposite. In this way cut-set determined by other edges of the spanning tree with thick edges can be shown to have non-negative weights.

Each edge of an MST is in the majority direction of the fundamental cutset it defines.

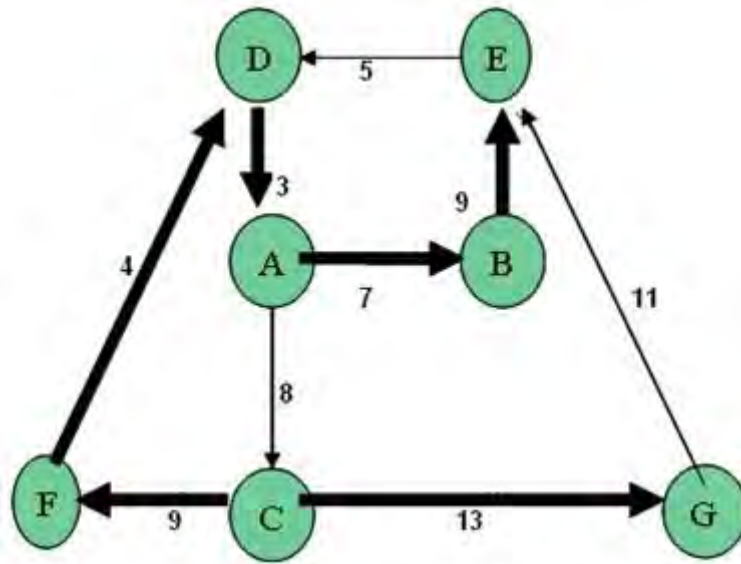


Figure 3.2: Majority Spanning Tree

## 3.2 Existing MST Algorithm For Round Robin Tournament Problem

### 3.2.1 Majority Spanning Tree Algorithm

In [19], Kaykobad et al. consider the problem of ranking players by the criterion of minimizing the number of violations (upsets).

Let  $R$  be a ranking and  $G_R = (V_R, A_R)$  be a subgraph of  $G = (V, A)$  such that  $V_R = V$  and  $A_R = \{(i, j) \mid \text{rank of player corresponding to vertex } j \text{ is immediately below player corresponding to vertex } i\}$ . It is obvious that  $G_R = (V_R, A_R)$  is a spanning tree of  $G$ , more accurately  $G_R = (V_R, A_R)$  is a Hamiltonian semi-path.

**Theorem 3.2.1.** Let  $R$  be any optimal ranking of a tournament represented by  $G = (V, A)$ . Then  $G_R = (V_R, A_R)$  is an MST of  $G$  [19].

**Proof.** Suppose that  $G_R = (V_R, A_R)$  does not correspond to an MST. Then there exists a violating cutset (a cutset is said to be violating if its weight is negative)  $[V', V \setminus V']$ . This means that the set of players corresponding to  $V'$  has lost more games to the remaining players than they have won from them. Therefore, if we rank players corresponding to  $V \setminus V'$  first, without changing their relative ranking, and then rank players corresponding to  $V'$ , the number of upsets will be decreased. Therefore, the result is true.  $\square$

Let  $G^{ij}(R)$  be the subgraph of  $G$  induced by the set of vertices corresponding to players ranked from  $i$  to  $j$ , and let  $G_{ij}^m(R)$  be the subgraph of  $G^{ij}(R)$  having the same set of vertices, and only those arcs that correspond to the results of matches between consecutively ranked players.

Theorem 3.2.2. For any optimal ranking  $R$  and  $1 \leq i \leq j \leq n$ ,  $G_{ij}^m(R)$  must be an MST of  $G^{ij}(R)$  [19].

Proof. The algorithm MST finds a  $G_{ij}^m(R)$  that is an MST of  $G$ . If it is not so, then the ranking can be improved by swapping the set of consecutively ranked players as has been noted in the proof of Theorem 3.2.1. A systematic search for a violating cutset is carried out by this algorithm through choosing subdigraphs induced by consecutively ranked players, and then checking all its possible cutsets. If no more violating cutset can be found for any of the subdigraphs  $G^{ij}(R)$  for all possible values of  $i, j$ , the MST algorithm stops. However, the quality of solution can still be improved, as is done in ARRANGE, by swapping players around a cut-set with 0 weight, which will not deteriorate the current solution.  $\square$

From Theorem 3.2.2 we have the following corollaries relating the properties of the MST algorithm with those of HP and ARRANGE algorithms.

Corollary 3.2.3. For any ranking  $R$  obtained by the MST algorithm,  $G_{ij}^m(R)$  is a directed Hamiltonian Path of  $G^{ij}(R)$  for  $1 \leq i \leq j \leq n$  [19].

Corollary 3.2.4. For any ranking  $R$  obtained by the MST algorithm,  $R$  cannot be improved by applying Hamiltonian Path or ARRANGE algorithms [19].

By virtue of Corollary 3.2.3 MST ranking cannot be improved by the HP algorithm, whereas if a ranking can be improved by ARRANGE algorithm then one can still find a violating cutset for the MST algorithm to get an improved solution. Hence, ARRANGE algorithm cannot improve any solution obtained by the MST algorithm.

Below we give some explanations of what the algorithm does, and the functions that are used in the algorithm.

- $\text{cutset}(i, k, j)$  – is the difference between the numbers of outgoing arcs from set  $(i, k)$  to set  $(k + 1, j)$  and outgoing arcs from set  $(k + 1, j)$  to set  $(i, j)$ , where set  $(i, j)$  is the set of vertices corresponding to players ranked from  $i$  to  $j$ .



- $\text{maxwin}(i, j)$ – is the maximum number of wins of a player in set  $(i, j)$ .
- $\text{pair}(i, j)$ – corresponds to an upset if the player ranked  $j$  defeats the player ranked  $i$ .
- $\text{size}$ – is the number of players in the tournament.

In the algorithm  $i$ -loop selects the first vertex of  $G^{ij}(R)$ ,  $j$ -loop selects the last vertex of  $G^{ij}(R)$ , whereas  $k$ -loop selects the position of cutset. In case of the absence of any violating cutset, players, around a cutset with zero weight, are swapped, just as is done in ARRANGE. However, if even such a swapping does not guarantee an improvement of the solution the subset of players containing player corresponding to  $\text{maxwin}(i, j)$  is given better ranking by swapping.

---

#### Algorithm 5 Majority Spanning Tree

---

```

1: Majority ( MST)
2:   repeat
3:     swap ← false
4:     for i = 1 to size – 1 do
5:       for j = i + 1 to size do
6:         for k = i to j – 1 do
7:           if cutset(i, k, j) < 0
8:             swap ← true
9:           else if cutset(i, k, j) = 0 then
10:            if pair(i, j) or (i – 1, k + 1) or (k, j + 1) is upset then
11:              swap ← true
12:              swap respective pair
13:            else
14:              if maxwin(i, k) < maxwin(k + 1, j)
15:                swap ← true
16:                swap respective pair
17:              endif
18:            endif
19:          endif

```

---

---

**Algorithm 5 Majority Spanning Tree Continued...**

---

```
20:         if swap = true then
21:             swap set ({i, k}, {k + 1, j})
22:             break i-loop
23:         endif
24:     endfor (k-loop)
25: endfor ( j-loop)
26: endfor (i-loop)
27: until not swap
```

---

Assuming the number of players in the tournament to be  $n$ , complexity of the MST algorithm can be derived as follows: In the  $k$ -loop, calculation of cutset value requires  $O(n)$  operations. Each of the  $i$ ,  $j$  and  $k$ -loop will be done at most  $n$  times for a single swap, which will reduce the number of violations by 1. The amount of computation for this is at most  $O(n^4)$ . Since there can be at most  $O(n^2)$  violations initially, the algorithm requires at most  $O(n^6)$  calculations.

### 3.2.2 A Modified Algorithm For Round-Robin Tournament

For modification of MST, in [11], Avijit et al. introduced the following functions:

- $kut(i,k)$ – cutset  $(i,k,n)$ , the difference between the numbers of outgoing arcs from set  $(i, k)$  to set  $(k + 1, n)$  and outgoing arcs from set  $(k + 1, n)$  to set  $(i, k)$ , where set  $(i, k)$  is the set of vertices corresponding to players ranked from  $i$  to  $k$ .
- $updateKut()$ – updates the  $kut(i,k)$  whenever the  $SetSwap(i,k, j)$  is done and it does this in the following incremental approach :

$$kut(i, k) = kut(i, k - 1) + \text{sum}$$

where,

$$\text{sum} = \sum_{l=k+1}^n \text{Arcs}(k, l) - \sum_{l=i}^{k-1} \text{Arcs}(l, k)$$

$$\text{Arcs}(k, l) = \begin{cases} 1, & \text{if player ranked } k \text{ defeats player ranked } l \\ -1, & \text{if player ranked } l \text{ defeats player ranked } k \\ 0, & \text{if } k=l \end{cases}$$

each of  $i$  and  $k$  loops is run at most  $n$  times, and computation of sum takes another  $O(n)$  complexity. So, the total complexity of  $\text{UpdateKut}()$  is  $O(n^3)$ . To derive the relationship between cutset  $(i, k, j)$  and  $\text{kut}(i, k)$ , we consider the set  $(i, n)$  as a union of 3 disjoint sets: set  $(i, k)$ , set  $(k + 1, j)$  and set  $(j + 1, n)$ .

Let,

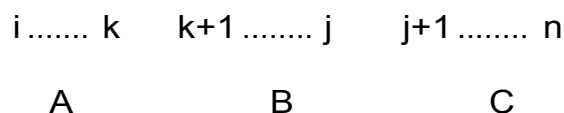
A denote the set  $(i, k)$

B denote the set  $(k + 1, j)$

C denote the set  $(j + 1, n)$

Then if we express  $AB = \text{cutset}(i, k, j)$

We get,  $A(B + C) = \text{cutset}(i, k, n) = \text{kut}(i, k)$



here,  $\text{cutset}(i, k, j) = AB$

$$= A(B + C) + BC - (A + B)C$$

$$= \text{kut}(i, k) + \text{kut}(k+1, j) - \text{kut}(i, j)$$

---

**Algorithm 6 Modified MST**

---

```
1:   Repeat
2:     done = false
3:     for i = 1 to n-1 do
4:       for j = i + 1 to n do
5:         for k = i to j-1 do
6:           if kut(i, k) + kut(k + 1, j) - kut(i, j) < 0 then
7:             done = true
8:             SetSwap(i, k, j)
9:             UpdateKut ()
10:            UpdateMaxWin ()
11:          else if kut(i, k) + kut(k + 1, j) - kut(i, j) = 0 then
12:            if maxwin(i, k) < maxwin(k + 1, j) then
13:              done = true
14:              SetSwap(i, k, j)
15:              UpdateKut ()
16:              UpdateMaxWin ()
17:            end if
18:          end if
19:        end for (k-loop )
20:      end for ( j-loop )
21:    end for (i-loop )
22:  Until not done
```

---

Assuming the number of players in the tournament to be  $n$ , complexity of the MST algorithm can be derived as follows: execution of UpdateKut() to update the kut ( $i, k$ ) value requires computation of at most  $O(n^3)$  complexity. Execution of UpdateMaxWin() to compute the maxwin( $i, j$ ) value requires computation of at most  $O(n^3)$  complexity. Each of the  $i, j$  and  $k$ -loop will be done at most  $n$  times for a single swap, which will reduce the number of violations by at least 1. So, the amount of computation is at most  $O(n^3)$ . Since there can be at most  $O(n^2)$  violations initially, the algorithm requires at most  $O(n^5)$  computation. Space requirement of kut is  $O(n^2)$ .

### 3.3 Limitations in Existing Algorithms

- Only consecutive set swap is possible.

For example if there are three set A, B and C, the algorithm can swap only set(A, B) or set(B,C) but can not swap set(A,C).

- Because of this reason most of the time required number of passes increases.

### 3.4 Improved MST Algorithm

In this Section, we propose an improved MST algorithm which gives better results compared to the MST algorithm and modified MST algorithm. We consider only simple connected digraphs  $G = (V, A)$ . Spanning trees of any digraph are denoted by  $T$ . A directed cutset  $(V_i, V_j)$  is defined as  $(V_i, V_j) = \{(k, l) \mid k \in V_i, l \in V_j\}$ . For improvement of the algorithm we divide the data in three sets and compare among them. In this algorithm we can swap sets which are not adjacent but in the previous algorithms this feature was not established. Below we give some explanations of what the algorithm does, and the functions that are used in the algorithm.

- $\text{cutset}(i, l, l + 1, k)$ – is the difference between the numbers of outgoing arcs from set  $(i, l)$  to set  $(l + 1, k)$  and outgoing arcs from set  $(l + 1, k)$  to set  $(i, l)$ .
- $\text{cutset}(l + 1, k, k + 1, j)$ – is the difference between the numbers of outgoing arcs from set  $(l + 1, k)$  to set  $(k + 1, j)$  and outgoing arcs from set  $(k + 1, j)$  to set  $(l + 1, k)$ .
- $\text{cutset}(i, l, k + 1, j)$ – is the difference between the numbers of outgoing arcs from set  $(i, l)$  to set  $(k + 1, j)$  and outgoing arcs from set  $(k + 1, j)$  to set  $(i, l)$ .

---

**Algorithm 7 Improved MST**

---

```
1: Repeat
2:   swap = false
3:   for i = 0 to less than size-2 do
4:     for j = i + 2 to less than size do
5:       for k = i to j do
6:         for l = i to k do
7:           if
8:             (cutset(i, l, l + 1, k) + cutset(l + 1, k, k + 1, j) + cutset(i, l, k + 1, j)) < 0
9:               swap (i, l, k + 1, j)
10:            elseif
11:              (cutset(i, l, l + 1, k) + cutset(l + 1, k, k + 1, j) + cutset(i, l, k + 1, j)) = 0
12:                if ( pair(i - 1, i) or pair(l, l + 1)
13:                  or pair(k, k + 1) or pair(j, j + 1) is upset) then
14:                    swap respective pair
15:                  elseif (maxwin(i, l) < maxwin(k + 1, j))
16:                    swap respective pair
17:                endifor l loop
18:              endifor k loop
19:            endifor j loop
20:          endifor i loop
21:        Until not swap
```

---

Assuming the number of players in the tournament to be  $n$ , complexity of the Improved MST algorithm can be derived as follows: In the l-loop, calculation of cutset value requires  $O(n)$  operations. Each of the  $i, j, k$  and  $l$ -loop will be done at most  $n$  times for a single swap, which will reduce the number of violations by 1. The amount of computation for this is at most  $O(n^5)$ . Since in the worst possible case there can be at most  $O(n^2)$  violations, then the algorithm requires at most  $O(n^7)$  calculations. In case no new ranking is formed, the algorithm requires  $O(n^5)$  steps.

### 3.4.1 Procedure of Improved MST

Figure 3.3 shows a tournament digraph with six players and the result of the game in matrix form. We use this digraph to explain the procedure of our algorithm. Let, at any instance the algorithm divides the graph into three sets I, J and K, Figure 3.4 shows this scenario. According to Improved MST procedure first calculate the value of  $\text{cutset}(I, J)$ ,  $\text{cutset}(I, K)$  and  $\text{cutset}(J, K)$ , then sum up these values and take further decisions.

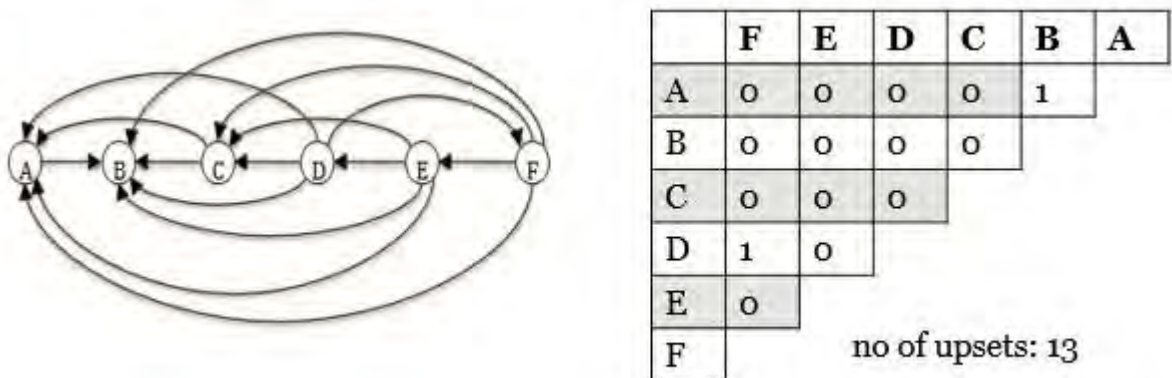


Figure 3.3: A Tournament digraph and a matrix of the result of the tournament.

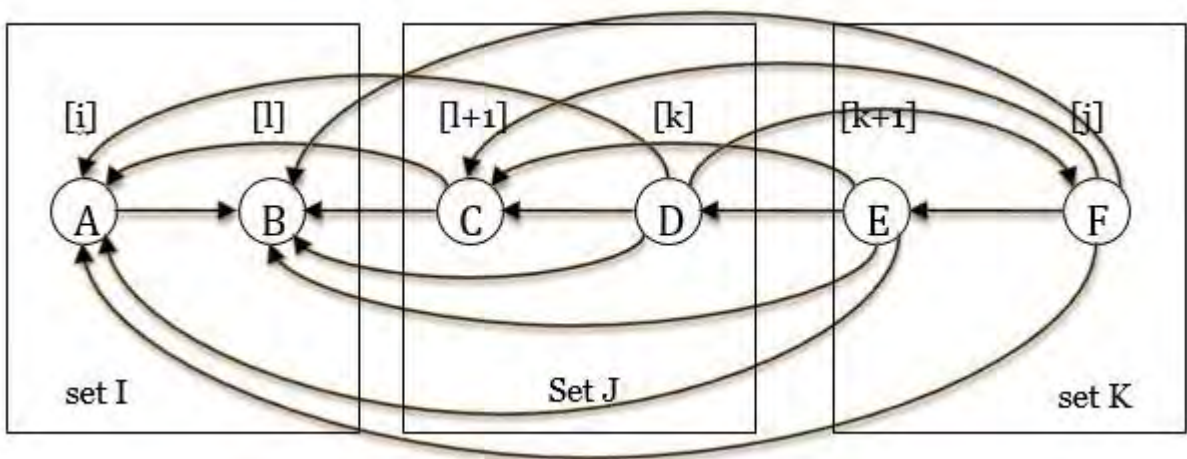


Figure 3.4: The algorithm divides the graph into three sets.

For this instance,  $\text{cutset}(I, J) = \text{cutset}(i, l, l + 1, k)$

$$= \text{set}(i, l) - \text{set}(l + 1, k)$$

$$= 0 - 4 = -4$$

$$\text{cutset}(I, K) = \text{cutset}(i, l, K + 1, j)$$

$$= \text{set}(i, l) - \text{set}(K + 1, j)$$

$$= 0 - 4 = -4$$

$$\text{cutset}(J, K) = \text{cutset}(l + 1, k, k + 1, j)$$

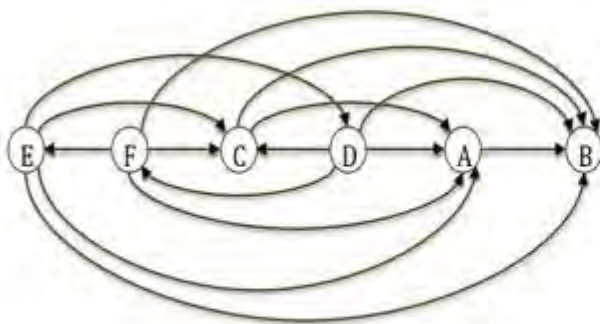
$$= \text{set}(l + 1, k) - \text{set}(k + 1, j) =$$

$$0 - 3 = -3$$

$$\text{cutset value} = \text{cutset}(I, J) + \text{cutset}(I, K) + \text{cutset}(J, K)$$

$$= -4 - 4 - 3 = -11$$

According to Improved MST condition if the summation of three cutsets is less than zero then the algorithm swap set I and set K. Figure 3.5 shows the situation after swapping.



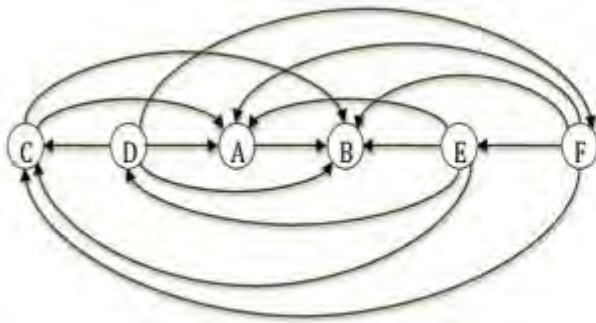
	<b>B</b>	<b>A</b>	<b>D</b>	<b>C</b>	<b>F</b>	<b>E</b>
<b>E</b>	1	1	1	1	0	
<b>F</b>	1	1	0	1		
<b>C</b>	1	1	0			
<b>D</b>	1	1				
<b>A</b>	1					
<b>B</b>						

no of upsets: 3

Figure 3.5: Situation of digraph after swapping set I and set K

Figure 3.5 reveals that the number of upset reduces by 10. For proving that most of the time nonconsecutive set swap is better than consecutive set swap we also swap consecutive set. First we swap set I and set J then calculate the upsets and found that number of upset is 9, Figure 3.6 shows this scenario. We also swap another two consecutive sets, set J and set K where we found that number of upset is 11. Our improved MST algorithm gives much better result comparing these cases.

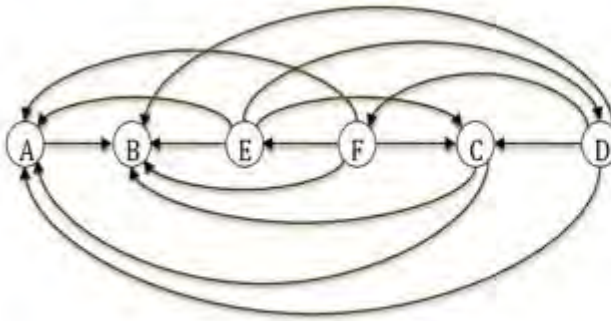




	F	E	B	A	D	C
C	0	0	1	1	0	
D	1	0	1	1		
A	0	0	1			
B	0	0				
E	0					
F						

no of upsets: 9

Figure 3.6: Situation of digraph after swapping set I and set J



	D	C	F	E	B	A
A	0	0	0	0	1	
B	0	0	0	0		
E	1	1	0			
F	0	1				
C	0					
D						

no of upsets: 11

Figure 3.7: Situation of digraph after swapping set J and set K

### 3.5 Hybrid Algorithm

We have fabricated a Hybrid algorithm by combining four algorithms which are previously used for solving round-robin tournament problem as well as our newly developed improved algorithm. The output of Hybrid algorithm produces a better result compared to any other algorithm. However, in the worst case scenario its complexity is quite high. In contrast, this algorithm is not time consuming for most cases because we sort the dataset in the beginning using sort algorithm. Afterwords, we improve the result by running the sorted output on HP. Complexity of sort and HP is negotiable, so we get a less violating dataset compared to initial input within a short period of time. As IMST uses a dataset which is already well ranked, so it does not require much effort on moving the sets of players, thus, it does not take much time. Hybrid algorithm keeps running until the amount of upset is declining.

Moreover when the result becomes stable it terminates the program.

---

Algorithm 8 Hybrid Algorithm

---

```
1: Sort the players according to number of wins
2: HplInput ← Sorted Result
3: Hamiltonian ()
   \ \ Works on the data stored on the HplInput
   \ \ Procedure is given on Algorithm 2
4: ArrangInput ← Result of Hamiltonian
5: Improvement ← true
6: while (Improvement) do
7:   BeforeUpset ← count upset before calling Arrange
8:   Arrange ()
   \ \ Works on the data stored on the ArrangInput
   \ \ Procedure is given on Algorithm 3
9:   AfterUpset ← count upset after calling Arrange
10:  if (AfterUpset ≥ BeforeUpset) then
11:    break
12:  end if
13: end while
14: MSTInput ← Final Result of Arrange
15: while (Improvement) do
16:   BeforeUpset ← count upset before calling MST
17:   MST ()
   \ \ Works on the data stored on the MSTInput
   \ \ Procedure is given on Algorithm 5
18:   AfterUpset ← count upset after calling MST
19:   if (AfterUpset ≥ BeforeUpset) then
20:     break
21:   end if
22: end while
```

---

---

**Algorithm 8 Hybrid Algorithm Continued...**

---

```
23: ImpMSTInput ← Final Result of MST
24: while (Improvement) do
25:   BeforeUpset ← count upset before calling Improved MST
26:   ImprovedMST ()
      \\ Works on the data stored on the ImpMSTInput
      \\ Procedure is given on Algorithm 7
27:   AfterUpset ← count upset after calling Improved MST
28:   if (AfterUpset ≥ BeforeUpset) then
29:     break
30:   end if
31: end while
32: Give final result
33: end Hybrid algorithm
```

---

The Improved MST and Hybrid algorithm discussed in this Chapter can substantially improve upon an existing ranking in terms of reducing the number of violations. As will be demonstrated in next Chapter, this algorithm is superior in terms of reducing violations to any of the heuristics presently available.

# Chapter 4

## Implementations and Experiments

To validate the correctness of our new algorithm, extensive simulation experiments have been conducted. We implement our proposed improved MST algorithm and previously developed MST algorithm. In addition, we developed a Hybrid algorithm where we implement Sort, Hamiltonian Path, Arrange and MST algorithms. Then compare the performance of these algorithms. In this Chapter, we describe the experimental setup and results in brief. In Section 4.1 we provide an overview of our implementation of Sort, HP, Arrange, MST and improved MST algorithm and Section 4.2 presents the simulation results to evaluate the performance of our proposed Improved MST algorithm and Hybrid algorithm in terms of both number of violations and computational time.

### 4.1 Implementation

This Section describes the implementation of Sort, HP, Arrange, MST and our proposed Improved MST algorithm. We developed a Hybrid algorithm in which we implement these algorithms as individual method. Here, we describe the different module of the implementation process and their functionality.

#### 4.1.1 Implementation of Sort

Inputs: The main parameters for the Sort algorithm are number of players(n) and the result of all the games they have played. These information is stored in a file in the form of

upper triangular matrix of 0 and 1. We read this data according to file name then stored this information in a 2D array named inputArray

Outputs: The output shows the number of upsets within players and their rank.

Scenario: Here we simply count the number of wins of each player then sort them according to the number of wins.

#### 4.1.2 Implementation of HP

Inputs: The main parameters for the HP algorithm are number of players( $n$ ) and the result of all the games they have played. Actually, this input is the output of Sort algorithm.

Outputs: The output shows the number of upsets within players and their rank.

Scenario: Find the violation of consecutive players among  $n$  players. Say, there exist a pair of players  $P_i$  and  $P_{i+1}$  where  $P_i$  is the  $i^{\text{th}}$  ( $0 < i < n$ ) player in the derived ranking, yet player  $P_{i+1}$  defeated player  $P_i$  in their match. Therefore, then new ranking will change to  $P_{i+1}, P_i$ .

#### 4.1.3 Implementation of Arrange

Inputs: The main parameters for the Arrange algorithm are number of players( $n$ ) and the result of all the games they have played. Actually, this input is the output of HP algorithm.

Outputs: The output shows the number of upsets within players and their rank.

Scenario: Starting with a given ranking  $R = (P_1, P_2, \dots, P_n)$  of  $n$  players in a tournament, an improvement in the ranking (reduction in the number of violations) can be attempted by determining whether or not each player is ranked in the "best" possible position. Specifically, a check can be made to determine whether or not the moving of a player to some new position in the ranking, while keeping all other players fixed in their respective positions, will reduce the number of violations.

#### 4.1.4 Implementation of MST

Inputs: The main parameters for the MST algorithm are number of players( $n$ ) and the result of all the games they have played. Actually, this input is the output of Arrange algorithm.

Outputs: The output shows the number of upsets within players and their rank.

Scenarios: According to the procedure of MST algorithm first we calculate the Cutset value. After that, depending on this value we take the decision of what to do next. Let the Cutset value between two sets of players is less than zero, in such case we just swap those sets. If the Cutset value between two sets of players is equal to zero then we check another two conditions and if one of these condition is true then we swap respective pair of players. If the Cutset value is greater than zero then there will be no change between the sets.

#### 4.1.5 Implementation of Improved MST

Inputs: The main parameters for the Improved MST algorithm are number of players(n) and the result of all the games they have played. Actually, this input is the output of the MST algorithm.

Outputs: The output shows the number of upsets within players and give their final ranking.

Scenario: According to the procedure of Improved MST algorithm first we calculate the three cutset value with four parameters. Then sum up these values and depending on this value we take the decision of what to do next. Let the summation of three sets of players is less than zero, in such case we swap first and last set of players keeping the second set of players fixed in their respective positions. If this summation value is equal to zero then we check another two conditions ( i. upset between consecutive pair ii. max winner between first last set ) and if one of these condition is true then we swap respective pair of players. If the sum is greater than zero then there will be no change between the sets.

## 4.2 Experiments

For comparing performance, three algorithms, namely MST, Improved MST and Hybrid MST have been considered. The basis for comparison of the above-mentioned heuristics is a set of randomly generated tournaments of sizes ranging from 10 to 100 players. All heuristics have been programmed in Java which is an object oriented programming language. Eclipse is an Integrated Development Environment (IDE) is used for writing all the algorithms. All the algorithms were run on core i3 machine with 4 GB RAM and the oper-

ating system was windows 10. All the datasets are generated by a built in function rand() which returns a pseudo-random number in the range of 0 to Total Player. Performance of the heuristics has been measured in terms of both violations and computational time. Table 4.1 depicts the result of MST, Improved MST and Hybrid algorithm in terms of number of upsets. Most of the time previously developed MST algorithm improves solutions compared to Sort, HP and Arrange algorithms [19]. Our Improved MST algorithm has an improved outcome when put into comparison with MST algorithm and in no case does it deteriorate the MST solution, thus, it would be futile to compare those algorithms with our Improved MST algorithm. In order to obtain better statics in number of violations, in Table 4.2 we showed average of 12 different datasets (10 random datasets and one for worst case, another for best case) of same size for all three algorithms. In Table 4.3 we measured average computational time of these three algorithms.

Table 4.1: Number of upsets for different algorithms

No.of Players	Initial Upset	MST	Improved MST	Hybrid
10	25	10	9	8
10	24	13	11	8
10	19	10	10	9
10	24	13	11	8
10	22	14	12	10
10	28	12	9	7
10	25	15	11	9
10	25	10	10	10
10	28	9	9	6
10	15	8	9	8
10	0	0	0	0
10	45	0	0	0
20	156	45	27	24
20	52	31	29	23
20	56	33	29	23
20	68	44	41	38

No.of Players	Initial Upset	MST	Improved MST	Hybrid
20	66	51	43	39
20	138	55	46	37
20	65	43	39	37
20	79	56	45	44
20	79	55	51	48
20	69	56	45	44
20	0	0	0	0
20	190	0	0	0
30	60	65	62	56
30	106	103	82	80
30	106	109	82	85
30	110	127	105	96
30	120	130	104	101
30	140	159	120	120
30	120	114	97	94
30	119	128	104	103
30	84	112	92	88
30	116	121	103	101
30	0	0	0	0
30	435	0	0	0
40	577	206	156	156
40	636	144	110	110
40	564	201	168	164
40	522	261	228	207
40	501	276	225	225
40	568	228	182	175
40	541	252	196	192
40	529	260	210	210
40	584	196	158	158
40	547	265	209	208



No.of Players	Initial Upset	MST	Improved MST	Hybrid
40	0	0	0	0
40	780	0	0	0
50	566	517	490	433
50	592	506	499	441
50	592	506	499	441
50	598	515	434	434
50	604	500	473	439
50	567	511	436	436
50	591	520	517	445
50	602	508	420	419
50	570	518	511	438
50	598	526	500	436
50	0	0	0	0
50	1225	0	0	0
60	895	746	682	656
60	908	741	719	640
60	921	733	683	633
60	886	747	638	638
60	885	717	627	627
60	903	735	730	645
60	900	741	626	626
60	916	754	628	628
60	923	774	730	653
60	900	764	688	652
60	0	0	0	0
60	1770	0	0	0
70	1195	1061	970	901
70	1235	1046	1033	916
70	1207	1033	973	891
70	1211	1075	1019	926

No.of Players	Initial Upset	MST	Improved MST	Hybrid
70	1195	1038	998	920
70	1207	1048	1014	901
70	1210	1016	951	867
70	1201	1069	1032	894
70	1211	1073	1072	925
70	1210	1066	993	896
70	0	0	0	0
70	2415	0	0	0
80	1556	1368	1319	1194
80	1554	1325	1280	1182
80	1568	1383	1338	1201
80	1567	1372	1343	1182
80	1560	1359	1327	1212
80	1551	1346	1223	1223
80	1552	1358	1308	1199
80	1551	1347	1279	1189
80	1564	1356	1324	1194
80	1551	1350	1261	1179
80	0	0	0	0
80	3160	0	0	0
90	1986	1755	1613	1562
90	2014	1724	1724	1724
90	1888	1685	1607	1601
90	1967	1744	1656	1661
90	1986	1755	1613	1562
90	2021	1745	1696	1501
90	1943	1698	1623	1604
90	1921	1667	1721	1512
90	1898	1732	1622	1562
90	2213	1824	1723	1667

No.of Players	Initial Upset	MST	Improved MST	Hybrid
90	0	0	0	0
90	4005	0	0	0
100	2472	2092	1991	1845
100	2478	1951	1786	1696
100	2556	1877	1802	1780
100	2567	2167	1886	1856
100	2321	1988	1778	1699
100	2456	1950	1802	1710
100	2510	2192	1978	1812
100	2331	1851	1756	1756
100	2675	2250	2021	1945
100	2451	1932	1845	1802
100	0	0	0	0
100	4950	0	0	0

From Table 4.1 we see that MST, Improved MST and Hybrid algorithm gives the same result for tiny dataset. Small dataset, those are constructed of 10 to 40 players, for those Hybrid algorithm and Improved MST algorithm gives a better result than MST. Being a dataset of 20 players, which have initial upsets 156, after applying MST, Improved MST and Hybrid we get 45, 27, 4 upsets respectively. For large dataset, the results are far better than MST in Improved MST. Although Hybrid algorithm provides a superior result in all cases, however, for large datasets, the diversity of result is significant.

Table 4.2: Average number of upsets for different algorithms

No.of Players	Average Initial Upset	MST	Improved MST	Hybrid
10	23.3333	9.5000	8.5000	6.9167
20	84.8333	39.0833	32.9167	29.7500
30	156.2727	106.1818	86.4545	84.9091
40	420.0036	190.7500	153.5000	153.0000
50	592.0833	427.2500	398.2500	364.6667

No.of Players	Average Initial Upset	MST	Improved MST	Hybrid
60	900.5833	621.0000	562.5833	535.6667
70	1208.0833	877.0833	838.4167	753.0833
80	1561.1667	1130.3333	1083.5000	996.3333
90	1986.8333	1444.0833	1383.1667	1329.6667
100	2480.5833	1687.5	1553.75	1491.75

Table 4.3: Average computational time of three algorithms in seconds

No.of Players	MST	Improved MST	Hybrid
10	2.5000	3.1200	2.0076
20	3.3303	5.5200	7.2200
30	5.2100	7.7508	8.3544
40	46.3385	10.4862	19.0594
50	155.0833	179.8263	50.8765
60	719.0833	1447.865	1065.9663
70	1040.4167	2922.9983	3251.2354
80	4842.25	5953.7083	6779.2027
90	5678.0098	7912.686	8597.393
100	6871.0371	8990.5471	9861.0023

Table 4.3 shows the average execution time of each algorithm for 12 datasets of same size. We calculate this time by using `currentTimeMillis()` methods which exists in `System` class of `java.lang` package. The `System.currentTimeMillis()` method returns the current time in milliseconds and the granularity of the value depends on the underlying operating system. Here we call this method two times for each algorithm: first one is just before the algorithm start and the second one is just after the algorithm ends. Then compute the difference between them for getting actual processing time of each algorithm. For large datasets it takes a lot of time in millisecond, that is why we converted milliseconds into seconds.

For better understanding how the rank changes among players we show the initial ranking and all the new ranking of each algorithms. Table 4.4 shows the rank for 10 players given

by MST, Improved MST and Hybrid algorithms.

Table 4.4: Ranking of various algorithms for 10 players

Rank	1	2	3	4	5	6	7	8	9	10
Initial Rank:	A	B	C	D	E	F	G	H	I	J
MST Rank:	G	E	H	B	C	D	J	F	I	A
Improved MST Rank:	B	E	G	H	A	C	D	I	J	F
Hybrid Rank:	B	G	E	H	C	D	J	I	F	A

#### 4.2.1 Graphical Analysis of the Results

For understanding more transparently upset's reduction trends, we represented all the algorithms result in graphical form. Figure 4.1 is constructed on the average of ten random datasets of same size starting from 10 players and ending at 100 players. Figure 4.2 is representing the worst case scenario. Where as Figure 4.3 represents the best case scenario, followed by Figure 4.4 representing average computational time.

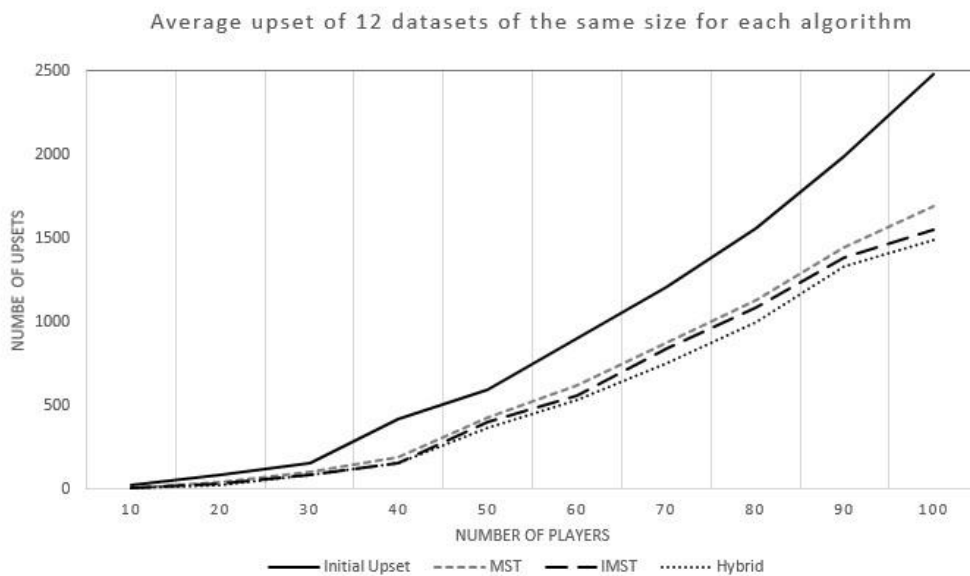


Figure 4.1: Comparison among MST, Improved MST and Hybrid Algorithm

In Figure 4.1, number of players are on X-axis and number of upset are shown on Y-axis. The graph reveals that the dataset is inclining, the amount of upset is also increasing. All

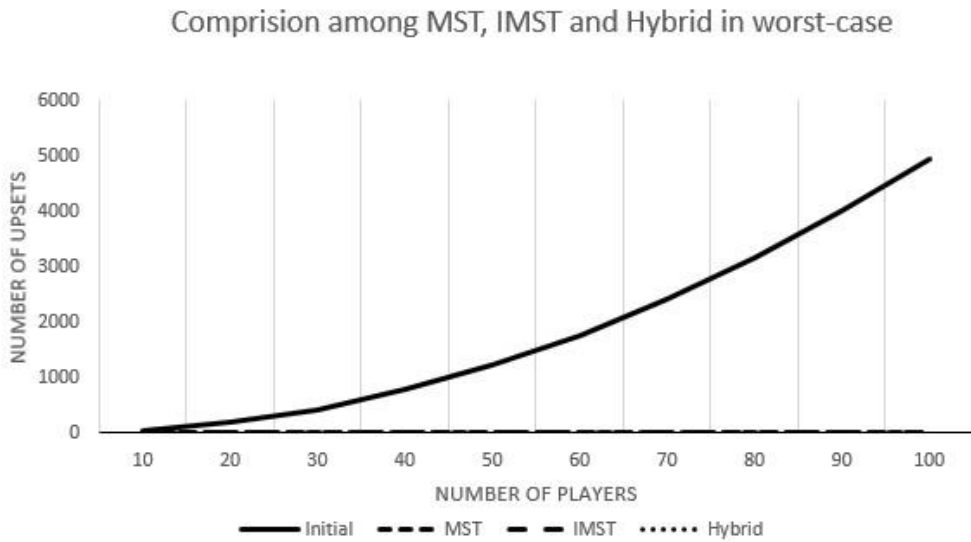


Figure 4.2: Situation in worst case

three algorithms almost give nearest result for 10 to 40 players. Afterwords, as the number of players increase, Improved MST and Hybrid algorithm give more enhancing results.

Figure 4.2 shows the worst case scenario where each player is defeated by all players below that specific rank. The input data that we have collected from worst case scenario is 0 in upper triangle. For this reason, in this case, initial upset is highest and after that when we pass this data in any algorithm we get 0 because of unique number of losses.

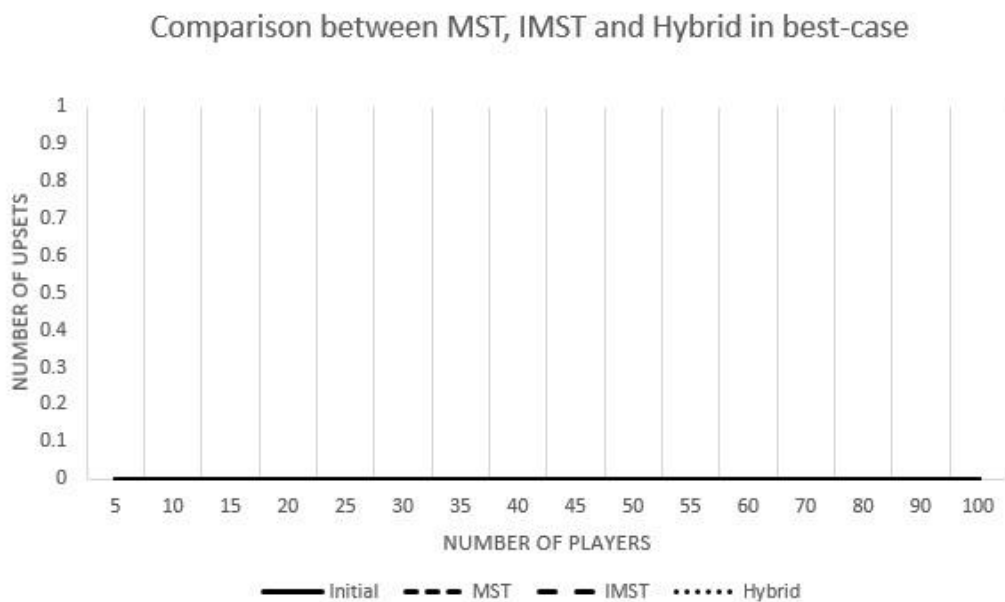


Figure 4.3: Situation in best case

Figure 4.3 shows the complete opposite scenario of Figure 4.2. Here, all the input is 1. This means there is no upset in the beginning and the algorithms have nothing to refuse.

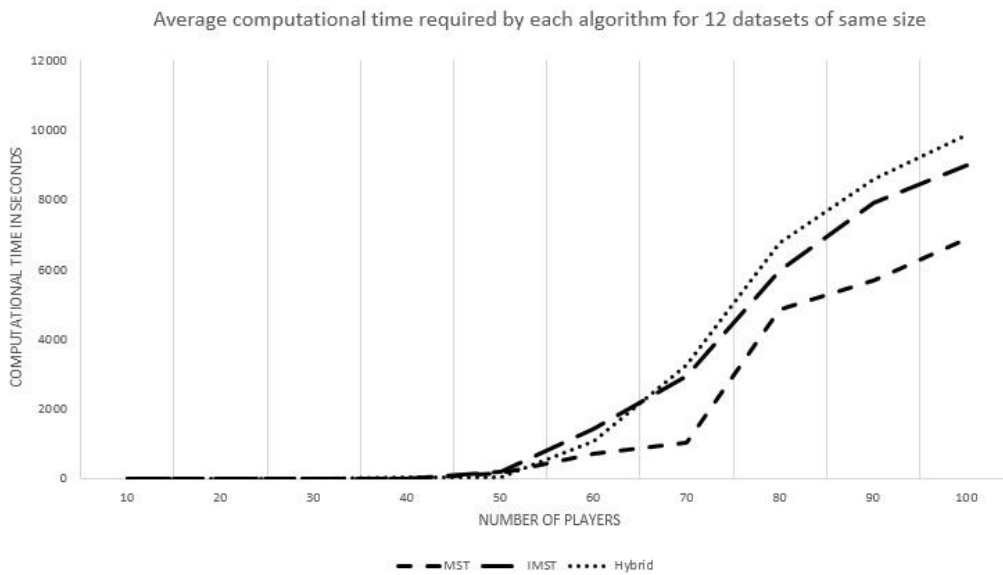


Figure 4.4: Average computational time required by each algorithm

Figure 4.4 represents the execution time of MST, Improved MST and Hybrid. This Figure reveals that execution time required for Hybrid is really high where as for MST that is very low. Although Hybrid and Improved MST give a much better result than MST.

# Chapter 5

## Conclusion

In this Chapter, we draw conclusion by highlighting the major contributions made in this thesis. We have also provided some directions for future research. In Section 5.1, the contribution of our thesis is elaborated. How this thesis can be extended in future is discussed in Section 5.2.

### 5.1 Contribution

At the time of starting of this thesis our first objective was to examine the properties of the round robin tournament problem structure and to analyze and understand the various Ranking algorithms. We elaborately discuss this problem as well as some other scenario where this problem arises. In this thesis we discuss about seven algorithms which are previously used for solving round-robin tournament problem: Iterated Kendall, Generalized Iterated Kendall, Hamiltonian Path, Arrange, Sort, MST and modified MST. Among them we also implement and run Sort, Hamiltonian Path, Arrange and MST.

Our second objective was to develop a new heuristic algorithm, which will somehow break the local minima and reach a solution, superior in terms of number of upsets. We have developed a new heuristic algorithm for improving ranking of the players named Improved MST algorithm. Furthermore, we have developed another algorithm named Hybrid in which we combine five algorithms: Sort, HP, Arrange, MST and Improved MST. A comparison is made of these two algorithms with previously developed MST algorithm on the basis of



randomly generated tournaments upto 100 players. We have also compared the average value of 3 different datasets of the same size. The statistics demonstrate the superiority of the Improved MST algorithm and Hybrid algorithm over existing heuristics in terms of the number of violations, although for very large problems (100 players or more e.g.) running times may become excessive.

## 5.2 Future Works

Minimum Feedback Arcset Problem is NP-hard even for tournament digraph probability finding a polynomial time algorithm is very small. For any problem of this complexity class since there is a computational explosion with the increase of the problem size, heuristic algorithms are used. However, quite often these heuristic algorithms are caught at a local minimum, and cannot come out. Sometimes, a significant amount of computational effort is needed to move out of such situation. In this thesis we have introduced a new heuristic to be known as Improved MST algorithm which somehow breaks comes out of the local minimum quite often. However, one needs to develop algorithms that will be more effective in solving problems of sizes that appear in practice.

## REFERENCES

- [1] I. Ali, W. D. Cook, and M. Kress. On the minimum violations ranking of a tournament. *Management Sci.*, (32):660-674, 1986.
- [2] V. Arya, N. Garg, R. Khandekar, A. Meyerson, K. Munagala, and V. Pandit. Local search heuristics for k-median and facility location problems. *SIAM Journal on Computing*, 33(3):544-562, 2004.
- [3] J. Aslam and M. Montague. Models for metasearch. *Proceedings of the Twenty-Fourth annual international ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 276-284, 2001.
- [4] N. Bansal, A. Blum, and S. Chawla. Correlation clustering. *Machine Learning*, 1(56):89-113, 2004.
- [5] R. Bao. Time relaxed round robin tournament and the nba scheduling problem. Technical report, Cleveland State University, 2009.
- [6] S. Chanas and P. Kobylanski. A new heuristic algorithm solving the linear ordering problem. *Computational Optimization and Applications*, 6(2):191-205, 1996.
- [7] P. Charbit, S. Thomasse, and A. Yeo. The minimum feedback arc set problem is np-hard for tournaments. *Combinatorics, Probability and Computing*, Cambridge University Press (CUP), 16:01-04, 2007.
- [8] T. Coleman and Wirth. Ranking tournaments: Local search and a new algorithm. *ACM J. Exp. Algor.*, Article 2.6(14), 2009.
- [9] W. Cook, I. Golan, and M. Kress. Heuristics for ranking players in a round-robin tournaments. *Computers Ops. Res.*, 15:135-144, 1988.
- [10] W. D. Cook and L. M. Seiford. Priority ranking and consensus formation. *Management Sci.*, 24(16):1721-1732, 1978.
- [11] A. Datta, M. Hossain, and M. Kaykobad. A modified algorithm for ranking players of a round-robin tournament. *International Journal of Computer Mathematics*, 85(1):1-7, 2007.

- [12] C. Dwork, R. Kumar, M. Naor, and D. Sivakumar. Rank aggregation revisited. Proceedings of the Tenth International World Wide Web Conference (WWW10), pages 613-622, 2001.
- [13] P. Eades, X. Lin, and W. Smyth. A fast and effective heuristic for the feedback arcset problem. 47(6):319-323, 1993.
- [14] P. Eades and N. C. Wormald. Edge crossings in drawings of bipartite graphs. *Algorithmica*, 11(4):379-403, 1994.
- [15] M. R. Garey and D. S. Johnson. *Computers and intractability: A guide to the theory of npcompleteness*. 1990.
- [16] M. Henz. Scheduling a major college basketball conference—revisited. Technical report, National University of Singapore, 2001.
- [17] D. B. Johnson. Finding all the elementary circuits of a directed graph. *SIAM Journal of Computing*, 4(1):77-84, 1975.
- [18] R. Karp. Reducibility among combinatorial problems. Proc. Sympos., IBM Thomas J. Watson Res.Center, Yorktown Heights, N.Y.:85-103, 1972.
- [19] M. Kaykobad, Q. Ahmed, A. Khalid, and R. Bakhtiar. A new algorithm for ranking players of a round-robin tournament. *Computers Ops. Res.*, 22(2):221-226, 1995.
- [20] M. Kendall. Further contributions to the theory of paired comparisons. *Biometrics*, 11:43-62, 1955.
- [21] C. Kenyon-Mathieu and W. Schudy. How to rank with few errors. Proceedings of the thirty-ninth annual ACM symposium on Theory of computing, (07):95-103, 2007.
- [22] D. J. Klein and M. Randic. Innate degree of freedom of a graph. *Journal of Computational Chemistry*, 4(8):516-521, 1987.
- [23] B. Massam. Spatial search: application to planning problems in the public sector. *Progress in Human Geography*, 6(4), 1982.
- [24] L. Pachter and P. Kim. Forcing matchings on square grids. *Discrete Mathematics*, 190:287-294, 1998.

- [25] G. Post and G. Woeginger. Sports tournaments, home-away assignments, and the break minimization problem. *Discrete Optimization*, 3(2):165-173, 2006.
- [26] R. V. Rasmussen and M. A. Trick. Round robin scheduling - a survey. *European Journal of Operational Research*, 188(3):617-636, 2007.
- [27] M. A. Trick. Integer and constraint programming approaches for round robin tournament scheduling. *Practice and Theory of Automated Timetabling IV*, 2740:63-77, 2002.
- [28] K. Wai and C. Ho. Rank aggregation for metasearch engines. ACM, New York, USA, (1581139128/04/0005):17-22, 2004.