Graph Factorization and Hamilton Path Based Balanced Tournament Design

Thesis

submitted in partial fulfillment of the requirements
for the award of degree of

Master of Engineering
in
Software Engineering

Submitted By

Desh Deepak Pathak
Roll No: 801131009

under the supervision of:

Mr. Ravinder Kumar
Assistant Professor(CSED)

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT
THAPAR UNIVERSITY
PATIALA - 147004

June 2013

Dedicated to
my parents
Smt. Damayanti Devi

Shri Vinod Pathak

Certificate

I hereby certify that the work which is being presented in the thesis entitled, "*Graph Factorization and Hamilton Path Based Tournament Design*", in partial fulfillment of the requirements for the award of degree of Master of Engineering in *Software Engineering* submitted in Computer Science and Engineering Department of Thapar University, Patiala, is an authentic record of my own work carried out under the supervision of *Mr. Ravinder Kumar* and refers other researchers work which are duly listed in the reference section.

The matter presented in the thesis has not been submitted for award of any other degree of this or any other University.

Signature:

(Desh Deepak Pathak)

This is to certify that the above statement made by the candidate is correct and true to the best of my knowledge.

(Mr. Ravinder Kumar)
Asstt. Professor
CSE Department

Countersigned by

(Dr. Maninder Singh)
Head
Computer Science and Engineering Department
Thapar University
Patiala

(Dr. S. K. Mohapatra)
Dean (Academic Affairs)
Thapar University
Patiala

Certificate

I hereby certify that the work which is being presented in the thesis entitled, "*Graph Factorization and Hamilton Path Based Tournament Design*", in partial fulfillment of the requirements for the award of degree of Master of Engineering in *Software Engineering* submitted in Computer Science and Engineering Department of Thapar University, Patiala, is an authentic record of my own work carried out under the supervision of *Mr. Ravinder Kumar* and refers other researchers work which are duly listed in the reference section.

The matter presented in the thesis has not been submitted for award of any other degree of this or any other University.

Desh Deepak Pathak
Signature:

(Desh Deepak Pathak)

This is to certify that the above statement made by the candidate is correct and true to the best of my knowledge.

(Mr. Ravinder Kumar)
Asstt. Professor
CSE Department

Countersigned by

(Dr. Maninder Singh)
Head
Computer Science and Engineering Department
Thapar University
Patiala

(Dr. S. K. Mohapatra)
Dean (Academic Affairs)
Thapar University
Patiala

Figure 1:

Acknowledgments

I express my sincere gratitude towards my guide **Mr. Ravinder Kumar** for his constant help, encouragement and inspiration throughout the project work. Without his invaluable guidance, this work would never have been a successful one. I would also like to thank my friends Abhishek Singh, Gaurav Rathi, Saurabh Jain and T.K. Pathak for their valuable suggestions and helpful discussions. Last, but not the least, I would like to thank the whole *Software Engineering* group which made my stay at Thapar University a memorable one.

Desh Deepak Pathak
Thapar University, Patiala
June 30, 2013

Abstract

A Hamilton path tournament design is based on round-robin tournament. For n teams, It takes $(n-1)$ days and each team plays in each stadium not more than twice. Moreover, the set of matches played in each stadium forms a Hamilton path. Formerly, an inductive proof has been given for the construction of Hamilton path tournament designs. It was shown for $n = 2^p \geq 8(p \geq 3)$. Here, I provide an algorithmic proof which constructs Hamilton path tournament designs for $n = 2^p \geq 8(p \geq 3)$ teams. It completes the inductive proof for practical means.

Contents

List of Figures

List of Tables

Chapter 1

Introduction: Tournament Designs

1.1 Round-robin Tournament

A round-robin tournament[1] is a tournament in which each team plays with every other team. When each team plays every other team once, it is known as single round-robin. Similarly, If each team plays all others double times, it is called a double round-robin tournament. Let n is the number of teams or players, a pure round robin tournament needs $(n/2) * (n-1)$ games. If n is even, then in each of $(n-1)$ rounds, $n/2$ games can be played in parallel and it requires $n/2$ stadiums for complete tournament. If n is odd, it will take n rounds, each with $(n-1)/2$ matches and one team will not play any match in that round. The round-robin algorithm can be stated by assigning each team a number and pair them off in the first round.

Example 1 Let for $n = 6$, team numbers are 1, 2, 3, 4, 5 and 6. The first round can be $\{(1,4),(2,5),(3,6)\}$. Now, fix one team and rotate the others clockwise one position. In this way, other rounds are $\{(1,5),(4,6),(2,3)\}$, $\{(1,6),(5,3),(4,2)\}$, $\{(1,3),(6,2),(5,4)\}$ and $\{(1,2),(3,4),(6,5)\}$. See Table 1.1.

1.1.1 Applications of Round-robin Tournament

In sports with a large number of competitive matches per season, double round-robins are common. Most association football leagues in the world are organized

Table 1.1: Round-robin schedule, for $n = 6$

	Stadium 1	Stadium 2	Stadium 3
Day 1	(1, 4)	(2, 5)	(3, 6)
Day 2	(1, 5)	(4, 6)	(2, 3)
Day 3	(1, 6)	(5, 3)	(4, 2)
Day 4	(1, 3)	(6, 2)	(5, 4)
Day 5	(1, 2)	(3, 4)	(6, 5)

on a double round-robin basis, in which every team plays all others in its league once at home and once away. This system is also used during qualification for major tournaments such as the FIFA World Cup[2] and the respective continental tournaments (e.g. UEFA European Championship[3], CONCACAF Gold Cup[4], etc.). There are also round-robin chess, go, and Scrabble tournaments. The World Chess Championship[5] decided in 2005 and in 2007 on an eight-player double round-robin tournament where each player faces every other player once as white and once as black.

Frequently, pool stages within a wider tournament are conducted on a round-robin basis. Examples with pure round-robin scheduling include the FIFA World Cup, UEFA European Football Championship and UEFA Cup (2004-2005) in football, Super Rugby (rugby union)[6] in the Southern Hemisphere during its past incarnations as Super 12 and Super 14, the Cricket World Cup[7], Indian Premier League(IPL)[8] Twenty-20 Cricket and many American Football college conferences, such as the Pacific-10. The group phase of the UEFA Champions League is contested as a double round-robin, as are most basketball leagues outside the United States, including the regular-season and Top 16 phases of the Euro league[9]; the United Football League[10] has used a double round-robin for both its 2009 and 2010 seasons.

1.2 Balanced Tournament Design

A balanced tournament design[1] for n teams is a round-robin schedule on $(n-1)$ days in which

- $n/2$ games, one in each stadium are played on each day, and
- each team uses each stadium at most twice.

A *balanced tournament design*[11] of order n, $BTD(n)$, defined a $2n$-set V is an arrangement of the $\binom{2n}{2}$ distinct unordered pairs of the elements of V into an $n * (2n-1)$ array such that:

1. every element of V is contained in precisely one cell of each column, and
2. every element of V is contained in at most two cells of any row.

An example of Balanced Tournament Design is shown in Table 1.2.

Table 1.2: Balanced Tournament designs, for $n = 6$

	Stadium 1	Stadium 2	Stadium 3
Day 1	(1, 2)	(3, 4)	(5, 6)
Day 2	(1, 3)	(2, 5)	(4, 6)
Day 3	(2, 6)	(3, 5)	(4, 1)
Day 4	(3, 6)	(4, 2)	(5, 1)
Day 5	(4, 5)	(1, 6)	(2, 3)

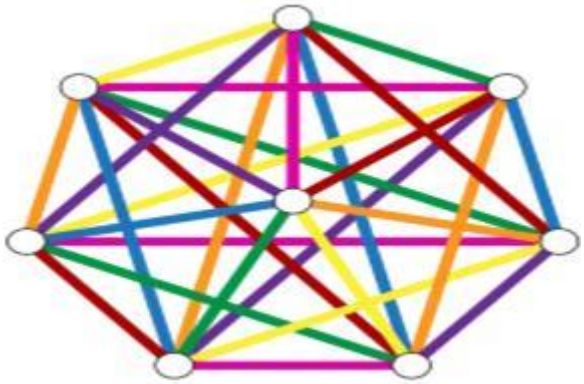

Figure 1.1: Complete Graph, K_8[13]

1.3 Graph Factorization and 1-factor

Before going in Graph Factorization and 1-factor in detail, It is necessary to focus on some basic graph terminology.

1.3.1 Complete Graph

A *complete graph*[12] is a graph with n vertices in which there is an edge between every two vertices. It has no loops and every two vertices share exactly one edge i.e. simple graph. We use the symbol K_n for a complete graph with n vertices. A complete graph K_8 is shown in Figure 1.1.

1.3.2 Chromatic Number and Index

Painting all the vertices of a graph with colors such that no two adjacent vertices have the same color is called the proper coloring of a graph. A graph G that requires k different colors for its proper coloring, and no less, is called a $k-$ *chromatic* graph, and the number k is called the *chromatic number*[12] of G. It deals with *vertex coloring* problem.

The minimum required number of colors for the edges of a given graph is called the *chromatic index*[14] of the graph. It deals with *edge coloring* problem. Edge coloring of an eight-vertex complete graph, see Figure 1.1. Each of the seven color classes consists of one edge from the center to a polygon vertex, together with the three perpendicular edges connecting pairs of polygon vertices.

1.3.3 Covering and Perfect Matching

In a graph G, a set g of edges is said to cover G if every vertex in G is incident on at least one edge in g. A set of edges that covers a graph G is said to be an edge covering, a covering subgraph, or simply a *covering*[12] of G. For example, a spanning tree in a connected graph is a covering. A Hamilton circuit in a graph is also a covering.

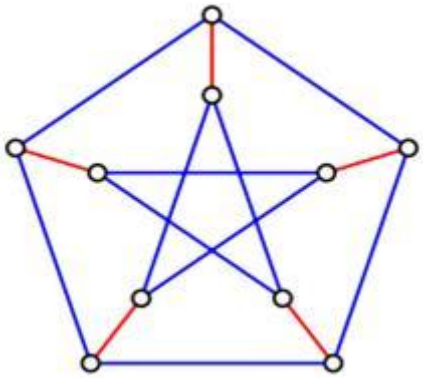

Figure 1.2: Petersen Graph [12]

A *covering* in which every vertex is of degree one is called a *dimer covering*[12]. A dimer covering is obviously a matching because no two edges in it are adjacent. Moreover, a dimer covering is a maximal matching. This is why a dimer covering is often referred to as a *perfect matching*.

1.3.4 Factorization of graph and 1-factor

A k-factor of a graph is a spanning k-regular sub graph, and a k-factorization partitions the edges of the graph into disjoint k-factors. A graph G is said to be k-factorable if it admits a k-factorization. In particular, a 1-factor is a perfect matching[15], and a 1-factorization of a k-regular graph is an edge coloring with k colors. If a graph is 1-factorable, then it has to be a regular graph. However, not all regular graphs are 1-factorable. A k-regular graph is 1-factorable if it has chromatic index k[15, 16]; examples of such graphs include:

- Any regular bipartite graph[15]. Hall's marriage theorem can be used to show that a k-regular bipartite graph contains a perfect matching. One can then remove the perfect matching to obtain a $(k-1)$-regular bipartite graph, and apply the same reasoning repeatedly, and
- Any complete graph with an even number of nodes[17].

1-factorization of K_8 is shown in Figure 1.1. Each set of edges with the same color is a 1-factor. However, there are also k-regular graphs that have chromatic index $k+1$, and these graphs are not 1-factorable; examples of such graphs include:

- Any regular graph with an odd number of nodes, and
- The Petersen graph, see Figure 1.2.

One can partition Petersen graph into a 1-factor (red color) and a 2-factor (blue color). However, the graph is not 1-factorable. See Figure 1.2.

1.4 Hamilton Path Tournament Design

For a given graph, a Hamilton path is a path in which each vertex appears exactly once. A Hamilton path tournament design[1, 18] involving n teams and $n/2$ stadiums, is a round robin schedule on $(n-1)$ days in which each team plays in each stadium at most twice, and the set of games played in each stadium induce a Hamilton path on n teams. A Hamilton path tournament design is a balanced tournament design in which for each stadium, the set of edges corresponding to games played in it form a Hamilton path on the set of teams.

1.5 Thesis Outline

As we have seen First chapter provides glimpse of tournament designs. It also covers basic graph terminology like Hamilton path, chromatic number, chromatic index, covering, matching and graph factorization. The rest of the thesis is outlined in this section. In Second chapter, literature reviews regarding graph factorizations and different tournament designs are discussed. In Third chapter, the problem statement is defined. Here, we focus on current existing system or scenario. Based on existing scenario, we define thesis objective and scope. Fourth chapter discusses the algorithmic proof of presence of Hamilton path based tournament for $n = 2^p \geq 8(p \geq 3)$, where n represents total number of teams. In this chapter we discuss use of permutation, combination and various file operations for achieving our objective. The intermediate and final results are also discussed in this chapter in form of different tables. The fifth chapter focuses on different test cases and outputs on various scenario of input value n. Finally, thesis ends with the conclusion and the scope for the future work in chapter six. The last three pages carry references and publications.

Chapter 2

Literature Review

The background of this whole research is based on graph theory concepts. Graph theory provides mathematical base to computer science algorithms. At first step of this work, we started with basics of graph theory. According to Narsingh Deo[19], a tree T is said to be a *spanning tree* of a connected graph G if T is a subgraph of G and T contains all vertices of G. He also defines a regular graph is a graph where every vertex has same degree. Also, a regular graph with vertices of degree k is called a *k-regular* graph. Further, Deo[12] explains that every tree with two or more vertices is 2-chromatic. Given a simple, connected graph G, partition all vertices of G into the smallest possible number of disjoint, independent sets. This problem is known as the *chromatic partitioning* of graphs. A *matching* in a graph is a subset of edges in which no two edges are adjacent. A single edge in a graph is obviously a matching. A *maximal* matching is a matching to which no edge in the graph can be added. For example, in a complete graph of three vertices any single edge is a maximal matching. The number of edges in a largest maximal matching is called the *matching number*.

According to Diestel and Reinharda[15], a set M of independent edges in a graph $G = (V, E)$ ia called a *matching*. M is a matching of $U \subseteq V$ if every vertex in U is incident with an edge in M. The vertices in U are then called matched(by M); vertices not incident with any edge of M are *unmatched*. A k-regular spanning subgraph of a graph G is called a k-factor. A 1-factor is also a perfect matching[12, 15]. Also, if a graph G accepts a k-factorization then it is called k-factorable. A k-factorization of graph G partitions all the edges of the graph into edge-disjoint k-factors. One can do 1-factorization of a k-regular graph by edge coloring with k colors. It means that if a k-regular graph has chromatic number k then it is 1-factorable. A regular bipartite graph[15] and a complete graph with even number of vertices[17] are examples of such graphs. 1-factorable graph must be a regular graph.

It is a well-known conjecture that if a regular graph G of order $2n$ has degree $d(G)$ satisfying $d(G) \geq n$, then G is the union of edge-disjoint 1-factors. It is well known that this conjecture is true for $d(G)$ equal to $2n-1$ or $2n-2$. Cetwynd and Hilton[20] has proved that it is true for $d(G)$ equal to $2n-3$, $2n-4$, or $2n-5$. They also shown that it is true for $d(G) \geq |V(G)|$. It means that if $k \geq 12n/7$ then given graph is 1-factorable. For a k-regular graph G, if total number of vertices n

is odd and $k \geq n$ then graph G is 1-factorable. Also, if n is even and $k \geq n-1$ then graph G is 1-factorable. It is called 1-factorization conjecture, introduced by Perkovic et al.[14].

J. H. Dinitz, P. Dukes and D. R. Stinson[21] used Steiner systems to arrive at uniform 1-factorizations of $K_{n,n}$. In their paper, they considered a weakening of the definitions of uniform and perfect 1-factorizations of the complete graph. Basically, they wanted to order the $(2n-1)$ 1-factors of a 1-factorization of the complete graph K_{2n} in such a way that the union of any two (cyclically) consecutive one-factors is always isomorphic to the same 2-regular graph. This property is termed *sequentially uniform*; if this 2-regular graph is a Hamiltonian cycle, then the property is termed sequentially perfect. They discussed several methods for constructing sequentially uniform and *sequentially perfect* 1-factorizations.

Many methods to compute 1-factorizations of a complete graphs of even order are presented. For complete graphs where the number of vertices is a power of 2, A. Prasant et al.[16] proposed several new methods to construct 1-factorizations. There methods are different from methods that make use of algebraic concepts such as Steiner triple systems, starters and all other existing methods. They also showed that certain complete multipartite graphs have 1-factorizations by presenting a method to compute 1-factorizations of such graphs. This method can be applied to obtain 1-factorizations of complete graphs with the number of vertices being a multiple of 4 or complete graphs with mn vertices provided a 1-factorization of K_m and a 1-factorization of K_n are known. Finally, deterministic and randomized back-tracking based algorithms to produce a 1-factorization for K_{2n} are presented by them. Both the algorithms always produce a 1-factorization if one exists.

For 1-factorization of a complete graph, a shift-rotate strategy (S-R)[16] has been given. In shift-rotate strategy for graph $K_{n,n}$, let the first end-points of each 1-factor are numbered as 1, 2, 3, ..., n in order. Similarly, the second end-points are numbered as $(n+1)$, $(n+2)$, ..., $2n$. Now, make pairs in order as $(1, n+1)$, $(2, n+2)$, ..., $(n, 2n)$. Further, for constructing the i_{th} 1-factor, shift and rotate the sequence $(n+1)$, $(n+2)$, ..., $2n$ by i positions to the left in order as $(1, n+1+i)$, $(2, n+i+2)$, ..., $(n, 2n+i-n)$. Each edge of graph $K_{n,n}$ comes out in one 1-factor. In general, an edge (i, j) with $1 \leq i \leq n$ and $(n+1) \leq j \leq 2n$, ordering of 1-factors the edge (i, j) comes out in the $(j-i+1)_{th}$ 1-factors as the end point of i.

Some survey has been done on Round-robin tournaments and applications. The round-robin schedule is used in many national and international tournaments. How round-robin and double round-robin is applicable can be understand by FIDE chess tournament[5] rules:

1. The drawing of lots for the first round of a round-robin tournament shall be arranged by the Chief Organizer, if possible, to be open to players, visitors and media. Responsibility for the actual pairings, including drawing of lots, rests with the Chief Arbiter.

2. The drawing of lots shall take place at least 12 hours (including one night) before the start of the first round. All participants should attend the cer-

emony of drawing of lots. A player who has not arrived on time for the drawing of lots may be included at the discretion of the Chief Arbiter. The first-round pairings shall be announced as soon as possible thereafter.

3. If a player withdraws, is excluded from a competition after the drawing of lots but before the beginning of the first round, or there are additional entries, the announced pairings shall remain unaltered. Additional pairings or changes may be made at the discretion of the Chief Arbiter in consultation with the players directly involved, but only if these minimise amendments to pairings that have already been announced.

4. The pairings for a round robin shall be made in accordance with the Berger tables, adjusted where necessary for double-round events.

5. If the pairings are to be restricted in any way-e.g. players from the same federation shall, if possible, not meet in the last three rounds - this shall be communicated to the players as soon as possible, but not later than the start of the first round.

6. For round-robin tournaments this restricted drawing of lots may be done by using the Varma tables, which can be used for tournaments of 9 to 24 players

Super Rugby[6] (also colloquially referred to as *Super*15) is the largest and pre-eminent professional rugby union football competition in the Southern Hemisphere. It uses round-robin schedule at group stages. The FIFA world cup[2] also uses round-robin schedule at group stages. The current final tournament features 32 national teams competing over a month in the host nation. There are two stages: the group stage followed by the knockout stage. Each group plays a round-robin tournament, in which each team is scheduled for three matches against other teams in the same group. The last round of matches of each group is scheduled at the same time to preserve fairness among all four teams. The top two teams from each group advance to the knockout stage. Points are used to rank the teams within a group. Since 1994, three points have been awarded for a win, one for a draw and none for a loss (before, winners received two points).

Gelling and Odeh[22] brought out Balanced tournament designs. For n teams, a round-robin tournament on $(n-1)$ days is a balanced tournament design in which $n/2$ matches are played on each day, one in each stadium and each team plays in each stadium not more than double times. J. H. Dinitz et al.[11] enumerated the balanced tournament designs on 10 points $BTD(5)$ and find that there are exactly $30,220,557$ non isomorphic designs. They also found that there are exactly two non–isomorphic partitioned $BTD(5)$'s and $8,081,114$ factored $BTD(5)$'s on 10 points. They enumerated other classes of balanced tournament designs on 10 points and also focused on combinatorial explosion phenomenon.

Schellenberg, Van Rees and Vanstone[23] proved that Balanced tournament designs exist for all even $n \geq 6$. Consequently, a Hamilton path tournament design is a Balanced tournament design. Hamilton path tournament designs are proved to exist for all even n not divisible by 4, 6 or 10[1, 18]. Recently, a schedule closely related to balanced tournament designs has been considered by

several authors. As in balanced tournament designs, we have n teams and $n/2$ stadiums, but in this case the goal is to create a schedule on n days in which each team pair plays at least once, each team uses each stadium exactly twice, and no two teams meet twice in the same stadium. Such schedules are called *stadium-balanced*. Note that each team has exactly one opponent which it meets twice. For any given balanced tournament design, one may create an n day schedule in which each team plays in each stadium exactly twice by adding for each stadium, a game between the two teams which play only once in it. However, the resulting schedule is not necessarily stadium-balanced, as the condition that no two teams meet twice in the same stadium may be violated. On the other hand, stadium-balanced schedules do not require the existence of a round consisting solely of games between teams meeting twice, hence we cannot always delete one round to create a balanced tournament design. Thus, balanced tournament designs and stadium-balanced schedules are not equivalent in the sense that one can be derived from the other by simple adjustments.

For even n, pairings in a round-robin tournament matches to a 1-factorization of a complete graph K_n with n vertices. Also, all games played on each day construct a 1-factor of complete graph K_n with n vertices. Yoshiko and Akihisa[1] gave a property $A(n)$ which is equivalent to existence of a Hamilton path tournament design.

Yoshiko and Akihisa proved that $A(n)$ is true for all $n = 2^p \geq 8(p \geq 3)$ by an inductive procedure. They defined a property $B(n)$ for inductive proof. In the property $B(n)$, $2K_n$ represents a complete graph K_n with n vertices, in which each edge is doubled. $A(2^p)$ exists consists of following three facts, first $B(4)$ holds, second $B(n)$ implies $A(2n)$ and third $B(n)$ implies $B(2n)$. One can verify the second facts by concerning Lemma 1[1] and Lemma 2[1]. They added that $A(n)$ may be alternatively be viewed as a statement on 1-factorizations and Hamilton path decompositions of K_n which are orthogonal to each other. The decomposition of the K_n into isomorphic subgraphs which are orthogonal is a subject on which there have been many studies.

Chengmin Wang, Jie Yan[24] demonstrated about the existence of near generalized balanced tournament designs. In their paper, they completed the existence of near generalized balanced tournament designs (NGBTDs) with block size 3 and 5. As an application, they obtained new classes of optimal constant composition codes.

Chapter 3

Problem Statement

3.1 Existing System

Yoshiko and Akihisa[1] defined a property $A(n)$ which is equivalent to existence of a Hamilton path tournament design.

Property $A(n)$[1] There exist $n/2$ Hamilton paths P_1, P_2, ..., $P_{n/2}$ and $(n-1)$ 1-factors F_1, F_2, ..., F_{n-1} of K_n satisfying

- (a1) the $n/2$ Hamilton paths P_1, P_2, ..., $P_{n/2}$ form a partition of $E(K_n)$,
- (a2) the $(n-1)$ 1-factors F_1, F_2, ..., F_{n-1} form a partition of $E(K_n)$, and
- (a3) $|P_i \cap F_j|$=1 ($i = 1$, ..., $n/2$, $j = 1$, ..., $n-1$).

They proved $A(n)$ is true for all $n = 2^p \geq 8 (p \geq 3)$ by an inductive procedure. For inductive proof, they defined a property $B(n)$. In the property $B(n)$, $2K_n$ represents a complete graph K_n with n vertices, in which each edge is doubled. In each pair of edge, one edge is assumed to be blue and other is red. Consequently, the set of blue and red edges form complete graph K_n separately. Further C_i, $i = 1, 2, ..., n-1$ denotes the subset of edges in which each node is incident to exactly two edges of C_i.

Property $B(n)$[1] There exist n Hamilton paths P_1, P_2, ..., P_n and $(n-1)$ 2-factors C_1, C_2, ..., C_{n-1} of $2K_n$ satisfying

- (b1) the $n/2$ Hamilton paths P_1, P_2, ..., $P_{n/2}$ form a partition of the blue edges of $E(2K_n)$,
- (b2) the $n/2$ Hamilton paths $P_{n/2+1}$, $P_{n/2+2}$, ..., P_n form a partition of the red edges of $E(2K_n)$,
- (b3) the $(n-1)$ 2-factors C_1, C_2, ..., C_{n-1} form a partition of $E(2K_n)$,
- (b4) $|P_i \cap C_j|$=1 ($i = 1, ..., n, j = 1, ..., n-1$).

- (b5) for any connected component D of any one of the $(n-1)$ 2-factors C_1, C_2, ..., C_{n-1}, the number of blue and red edges contained in D are equal and even (and thus the total number of edges is a multiple of four).

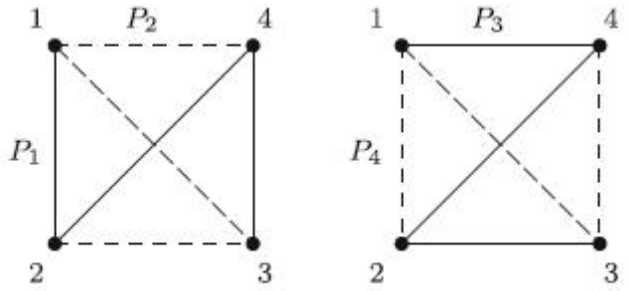

Figure 3.1: Hamilton paths for $B(4)$[1]

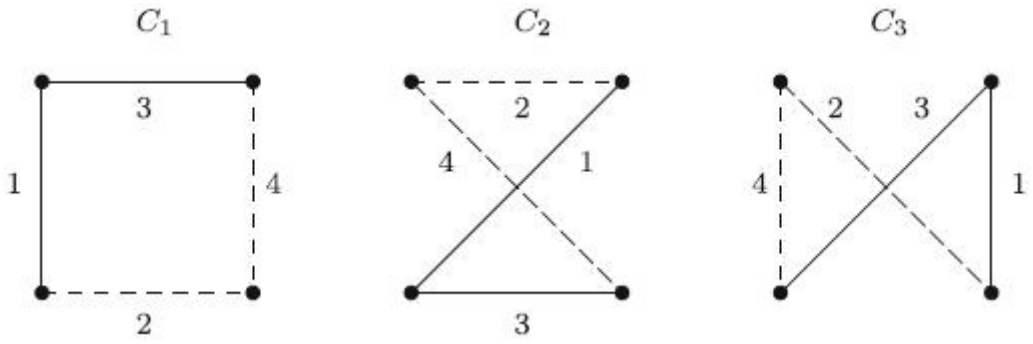

Figure 3.2: Three 2-factors for $B(4)$[1]

The existence of Hamilton paths and 2-factors in $B(4)$ are shown in Figure 3.1 and Figure 3.2. In Figure 3.1 the blue paths P_1 and P_2 are represented by solid and dotted lines respectively. Similarly, in Figure 3.2 the red paths are depicted by solid and dotted lines respectively. The three 2-factors are shown in Figure 3.2, the numbers on each edge denotes the Hamilton path to which it belongs to. Further, 2-factors and Hamilton path fulfill ($b1$) through ($b5$). $A(2^p)$ exists consists of following three facts:

1. $B(4)$ holds,
2. $B(n)$ implies $A(2n)$, and
3. $B(n)$ implies $B(2n)$.

Note that the whole work in existing system is done in form of mathematical induction proof. It can not be used as practical point of view. Also, It has no result sets.

3.2 Problem Statement

This work aims at improving the mathematical induction proof of Hamilton path based tournament to next practical level. The whole work covers the algorithmic proof of presence of Hamilton path in each stadium of balanced tournament. The main focus of this work is to present the implementation of Hamilton path based tournament designs for $n = 2^p \geq 8(p \geq 3)$, where n represents total number of teams.

3.3 Methodology used for solving the problem

The complete work is implemented and verified with results on *Turbo C++ IDE* using *C* language. The scope of this work is very wide and can be extended at simulator level. The following methodology and steps have been used in achieving the objective of this thesis:

- Formation of input data using ASCII codes. [Section 4.1]
- Permutation operation for generating set of 1-factors. [Section 4.1]
- Use of basic file operations like read, write, update, set-position, get-position using file pointers. In this step, we generate unique permutation and eliminates duplicates using *first step filter*. [Section 4.2]
- Combination generation and elimination of duplicate combination that not satisfying Balanced tournament. It uses *second step filter*. [Section 4.3]
- Filtration from input data based on criteria conditions in each step.
- Passing unique sets for Balanced tournament determination. [Section 4.4]
- Use of backtracking for Hamilton path determination. It takes data sets that satisfy Balanced Tournament criteria. [Section 4.5]
- Analysis and description of intermediate and final results in form of Tables in each step.

Chapter 4

Proposed Algorithmic Proof for Hamilton path based tournament design

4.1 Initialization and permutation

Let integer variable n represents number of teams or total number of nodes in a complete graph. First of all, n is tested for even value determination. After that, a character array r of size $n+1$ is initialized with character set {'0', '1', '2', '3', '4', ..., 'n'} i.e ASCII codes 48, 49, 50, 51, 52, ..., $n+48$, see Algorithm 1.

Algorithm 1 Initialization

```
1: procedure Initial(r, n)
2: begin
3: k:=48
4: for i := 1 to n + 1 do
5:     begin
6:     r[i] := k
7:     k := k + 1
8:     end
9: end
```

Algorithm 2 would be called with arguments as Permutate-Onefact(r, 2, n)[25]. It generates all permutations of character array r with last n elements. The first element, i.e r[1]='0', remains unchanged throughout permutation procedure. All permutations including first entry are written in a text file *permutate.txt* using file pointer fq and file operation File-Write(r, fq), see line 5 in Algorithm 2. The results for value $n = 4$ has been presented in Table 4.1. The same kind of results are obtained for other higher value of n like $n = 6$, $n = 8$ etc. If we partition each permutation in pairs then it represents a set of 1-factor(i.e for permutation 1234, a set of 1-factor would be (1, 2), (3, 4)).

Algorithm 2 Permutation Of One factors [25]

```
1: procedure Permutate-Onefact(r, k, n)
2: begin
3: if k = n + 1 then
4:     begin
5:     File-Write(r, fq)
6:     end
7: else
8:     begin
9:     for i := k to n + 1 do
10:        begin
11:        t1:=r[k]
12:        r[k]:=r[i]
13:        r[i]:=t1
14:        Permutate-Onefact(r, k + 1, n)
15:        t1:=r[k]
16:        r[k]:=r[i]
17:        r[i]:=t1
18:        end
19:    end
20: end
```

Table 4.1: All Permutations, for $n = 4$

0 1234	0 1243	0 1324	0 1342	0 1432	0 1423
0 2134	0 2143	0 2314	0 2341	0 2431	0 2413
0 3214	0 3241	0 3124	0 3142	0 3412	0 3421
0 4231	0 4213	0 4321	0 4312	0 4132	0 4123

4.2 First Step Filter

Now, just notice the entries 1234, 1243, 2134 and 2134 of the Table 4.1, if we partition all these permutations then it all form same set of 1-factors i.e (1, 2), (3, 4). These entries can be termed as duplicates. The same case occurs with other higher values of n like $n = 6, 8$ etc. For removing all such duplicates Algorithm 3 is given. It receives text file *permutate.txt* as input and writes first occurrence each entry in separate file say *uniqpermutate.txt*, see line 9 in Algorithm 3. Moreover, all other duplicates are identified and its first character '0' is replaced by '1', see Table 4.2.

Except read and write, there are two other file operations used in Algorithm 3 which are File-Pos(fq) and File-Set(fq, *pos*, 0), see line 6, 13, 15, 29, 31 and 35. The File-Pos(fq) operation tells current position of file pointer and File-Set(fq, *pos*, 0) operation sets file pointer at position *pos* from beginning. A variable f used in different algorithms throughout paper represents factorial of value n.

Algorithm 3 eliminating duplicates present in permutations based on un-ordered pair

```
1: procedure Duplicate-Filter1(f, n)
2: begin
3: for i := 1 to f do
4:     begin
5:     File-Read(p, n + 1, fq)
6:     pos2 :=File-Pos(fq)
7:     if p[1] = '0' then
8:         begin
9:         File-Write(p, fr)
10:        for k := i + 1 to f do
11:            begin
12:            c := 0
13:            pos :=File-Pos(fq)
14:            File-Read(q, n + 1, fq)
15:            pos1 :=File-Pos(fq)
16:            if q[1] = '0' then
17:                begin
18:                for j := 2 to n do
19:                    begin
20:                    if { ((p[j] = q[j])AND(p[j + 1] = q[j + 1]))OR
                           { ((p[j] = q[j + 1])AND(p[j + 1] = q[j])) then
21:                        begin
22:                        c := c + 1
23:                        end
24:                    j := j + 2
25:                    end
26:                if c=n/2 then
27:                    begin
28:                    q[1] := '1'
29:                    File-Set(fq, pos, 0)
30:                    File-Write(q[1], fq)
31:                    File-Set(fq, pos1, 0)
32:                    end
33:                end
34:            end
35:        File-Set(fq, pos2, 0)
36:        end
37:    end
38: end
```

Table 4.2: All Permutations after first step filter(duplicates detection), for $n = 4$

0 1234	1 1243	0 1324	1 1342	0 1432	1 1423
1 2134	1 2143	0 2314	1 2341	0 2431	1 2413
1 3214	1 3241	1 3124	1 3142	0 3412	1 3421
1 4231	1 4213	1 4321	1 4312	1 4132	1 4123

The entries of file *uniqpermutate.txt* for $n = 4$ is presented in Table 4.3. For $n = 4$, there are total six unique entries. As we have to consider only unordered pair, one can formulate what are total numbers of unique entries after applying first step filter i.e Algorithm 3 as follows:

- $n = 4$, total unique entry = 4!/(2!*2!)= 6 [see Table 4.3]
- $n = 6$, total unique entry = 6!/(2!*2!*2!)= 90 [see Table 4.4]
- $n = 8$, total unique entry = 8!/(2!*2!*2!*2!)= 2520 and so on.

This calculation is done in Algorithm 4 as final value of t.

Table 4.3: Unique Permutations after first step filter, for $n = 4$

1	2	3	4	5	6
0 1234	0 1324	0 1432	0 2314	0 2431	0 3412

Table 4.4: Unique Permutations after first step filter, for $n = 6$

1	0 123456	2	0 123546	3	0 123654	4	0 124536	5	0 124653	6	0 125634
7	0 132456	8	0 132546	9	0 132654	10	0 134526	11	0 134652	12	0 135624
13	0 143256	14	0 143526	15	0 143652	16	0 142536	17	0 142653	18	0 145632
19	0 153426	20	0 153246	21	0 153624	22	0 154236	23	0 154623	24	0 152634
25	0 163452	26	0 163542	27	0 163254	28	0 164532	29	0 164253	30	0 165234
31	0 231456	32	0 231546	33	0 231654	34	0 234516	35	0 234651	36	0 235614
37	0 243156	38	0 243516	39	0 243651	40	0 241536	41	0 241653	42	0 245631
43	0 253416	44	0 253146	45	0 253614	46	0 254136	47	0 254613	48	0 251634
49	0 263451	50	0 263541	51	0 263154	52	0 264531	53	0 264153	54	0 265134
55	0 341256	56	0 341526	57	0 341652	58	0 342516	59	0 342651	60	0 345612
61	0 351426	62	0 351246	63	0 351624	64	0 354216	65	0 354621	66	0 352614
67	0 361452	68	0 361542	69	0 361254	70	0 364512	71	0 364251	72	0 365214
73	0 453126	74	0 453216	75	0 453621	76	0 451236	77	0 451623	78	0 452631
79	0 463152	80	0 463512	81	0 463251	82	0 461532	83	0 461253	84	0 465231
85	0 563412	86	0 563142	87	0 563214	88	0 564132	89	0 564213	90	0 561234

4.3 Combination Calculation and Second Step Filter

In this section, we are about to perform basic combination operation on text file *uniqpermutate.txt*. Consider Algorithm 4 i.e *Perm-Store()*, it performs storage of unique permutations in 2-D character array *unp*[][]. It receives n, factorial f of n and file pointer fr of text file *uniqpermutate.txt*. Further, it calculates total unique permutation in variable t and at the end stores all unique permutation inside global 2-D character array *unp*[][] of size $t * n$. A chunk of $n + 2$ size is read in 1-D array r, see *File-read* operation in line 11. The value of total unique permutations t is also calculated in this algorithm. The different values of t for $n = 4$, 6 and 8 are 6, 90 and 2520 respectively.

Algorithm 4 Storage of unique permutations in 2-D character array *unp*[][]

```
 1: procedure Perm-Store(n, fr, f)
 2: begin
 3: t := f
 4: for i := 2 to n do
 5:     begin
 6:     t := t/2
 7:     i := i + 2
 8:     end
 9: for i := 1 to t do
10:     begin
11:     File-Read(p, n + 2, fr)
12:     for j := 1 to n do
13:         begin
14:         unp[i][j] := p[j + 2]
15:         end
16:     end
17: end
```

Once unique permutations are stored in array *unp*[][], Algorithm 5 named as *Duplicate-Filter2()* comes in action. It performs elimination of duplicates present in combinations that do not satisfy balanced tournament. It is presented in two parts due to large size. The part 2 is mentioned as Algorithm 6. It starts with initialization of 1-D integer array *comb*[], first initialization of array *comb*[] represents first combination out of $\binom{t}{n-1}$. In our problem we are dealing with a tournament of $(n - 1)$ days and $n/2$ teams each day. Just notice $repeat - until$ loop, at each iteration all entries of *comb*[] array are used as index for *unp*[][] array and corresponding entries of *unp*[][] array are assigned to array a[][]. *Next-Comb()* i.e Algorithm 7[26] generates next set of combination for array *comb*[]. It is called at the end of $repeat - until$ loop in Algorithm 6. The duplicate filtration condition is evaluated in *if* block in line 29 in Algorithm 5. At a time, all entries of array a[][] form a tournament of $(n - 1)$ days and $(n/2)$ teams play at each day. At the end of previous section, we have total t unique set of 1-factors for each even n.

Thats why, we need to consider total $\binom{t}{n-1}$ combinations for each set of $\{t, n\}$. If $n = 4$ and $t = 6$ then $\binom{6}{3}$=20, see Table 4.5 for all combinations.

Table 4.5: Total Combinations, for $n = 4$ and $t = 6$

123	124	125	126	134	135	136	145	146	156
234	235	236	245	246	256	345	346	356	456

There might be possibility of duplicates at this stage. Now, let us move to check availability of duplicates at this stage. It can be identified by some output analysis. Refer Table 4.6, 2-D array $a[][]$ containing duplicates in tournament with combination=125, for $n = 4$ and $t = 6$. It contains total 20 entries of array $a[][]$ for combination {1, 2, 5}. In 2nd and 3rd row of Table 4.6, there are same pairs (2, 4) and (1, 3). This is not a valid input for balanced tournament. Such kind of combinations are many in numbers for different n. In Algorithm 5 starting from line 19 to line 41, detection and elimination of invalid combinations have been done. Table 4.7 contains valid combinations for $n = 4$ after second step filter i.e Algorithm 5 part 1 and part 2 i.e Algorithm 6. Same scenario happens with higher value of n. As we notice at the end of part2 of Algorithm 5, a procedure *BTD(a, n)* is called. *BTD(a, n)* i.e Algorithm 8 is called only for valid combinations inside $repeat-until$ loop of Algorithm 5. It performs balanced tournament detection. It is called at each iteration of loop when conditions are satisfied. Now, it is time to recall balanced tournament definitions and criteria conditions before moving next sections. As we defined balanced tournament in chapter 1:

A balanced tournament design[1, 11] for n teams is a round robin schedule on $n - 1$ days in which

- $n/2$ games, one in each stadium are played on each day, and
- each team uses each stadium at most twice.

Based on this definition an algorithm for balanced tournament determination is given, see Algorithm 8 in next section.

Table 4.6: 2-D array $a[][]$ containing duplicates in tournament with combination=1 2 5, for $n = 4$ and $t = 6$

Combination No.	Stadium 1	Stadium 2
1	(1, 2)	(3, 4)
2	(1, 3)	(2, 4)
5	(2, 4)	(3, 1)

Table 4.7: Unique Combinations after second step filter, for $n = 4$ and $t = 6$

123	124	135	145	236	246	356	456

Algorithm 5 eliminating duplicates present in combinations that do not satisfy Balanced Tournament part 1

```
1: procedure Duplicate-Filter2(f, n)
2: begin
3: for i := 1 to n − 1 do
4:     begin
5:     comb[i] := i
6:     end
7: repeat
8:     begin
9:     for i := 1 to n do
10:        begin
11:        k := 1
12:        for j := 1 to n/2 do
13:            begin
14:            a[i][j].u = unp[comb[i]][k]
15:            a[i][j].v = unp[comb[i]][k + 1]
16:            k := k + 2
17:            end
18:        end
19:    for m := 2 to n − 1 do
20:        begin
21:        c := 0
22:        for i := 1 to m − 1 do
23:            begin
24:            l := 0
25:            for k := 1 to n/2 do
26:                begin
27:                for j := 1 to n/2 do
28:                    begin
29:                    if { ((a[m][k].u = a[i][j].u)AND(a[m][k].v = a[i][j].v))OR
                           ((a[m][k].u = a[i][j].v)AND(a[m][k].v = a[i][j].u))
   then
30:                        begin
31:                        l := l + 1
32:                        end
33:                    end
34:                end
35:            if l ≥ 1 then
36:                begin
37:                c := 1
38:                break
39:                end
40:            end
41:        end
```

Algorithm 6 eliminating duplicates present in combinations that do not satisfy Balanced Tournament part 2

```
42:         if c = 0 then
43:             begin
44:             BTD(a,n)
45:             end
46:     end
47:     until Next − Comb(comb, t, n − 1) = true
48: end
```

The concept of backtracking[27] and Hamilton path based tournament detection is used in later sections. Here, brief background on those terms are presented. Many problems are difficult to solve algorithmically. Backtracking makes it possible to solve at least some large instances of difficult combinatorial problems. The principal idea is to construct solutions one component at a time and evaluate such partially constructed candidates as. If a partially constructed solution can be developed further without violating the problems constraints, it is done by taking the first remaining legitimate option for the next component. If there is no legitimate option for the next component, no alternatives for any remaining component need to be considered. In this case, the algorithm backtracks to replace the last component of the partially constructed solution with its next option. It is convenient to implement this kind of processing by constructing a tree of choices being made, called the state-space tree. Its root represents an initial state before the search for a solution begins. The nodes of the first level in the tree represent the choices made for the first component of a solution; the nodes of the second level represent the choices for the second component, and so on. A node in a state-space tree is said to be promising if it corresponds to a partially constructed solution that may still lead to a complete solution; otherwise, it is called non-promising. Leaves represent either non-promising dead ends or complete solutions found by the algorithm.

As we discussed in chapter 1, for a given graph, a Hamilton path is a path in which each vertex appears exactly once. A Hamilton path tournament design involving n teams and $n/2$ stadiums, is a round robin schedule on $(n-1)$ days in which each team plays in each stadium at most twice, and the set of games played in each stadium induces a Hamilton path on n teams. As in balanced tournament designs, we have n teams and $n/2$ stadiums, but in this case the goal is to create a schedule on n days in which each team pair plays at least once, each team uses each stadium exactly twice, and no two teams meet twice in the same stadium. Such schedules are called *stadium-balanced*. Note that each team has exactly one opponent which it meets twice. For any given balanced tournament design, one may create an n day schedule in which each team plays in each stadium exactly twice by adding for each stadium, a game between the two teams which play only once in it. However, the resulting schedule is not necessarily stadium-balanced, as the condition that no two teams meet twice in the same stadium may be violated.

Algorithm 7 Combination Calculation [26]

```
1: procedure Next-Comb(ar, n, k)
2: begin
3: finished := 0
4: changed := 0
5: if k > 0 then
6:     begin
7:     i := k − 1
8:     while finished=0 AND changed=0 do
9:         begin
10:        if ar[i] < (n − 1) − (k − 1) + i then
11:            begin
12:            ar[i] := ar[i] + 1
13:            if i < k − 1 then
14:                begin
15:                for j := i + 1 to k − 1 do
16:                    begin
17:                    ar[j] := ar[j − 1] + 1
18:                    end
19:                end
20:            changed := 1
21:            end
22:        if i = 0 then
23:            begin
24:            finished := 1
25:            end
26:        i := i − 1
27:        end
28:    if changed = 0 then
29:        begin
30:        for i := 1 to k do
31:            begin
32:            ar[i] = i
33:            end
34:        end
35:    end
36: return changed
37: end
```

4.4 Balanced Tournament Determination

Balanced Tournament determination is done in Algorithm 8. It takes value n and 2-D array $a[][]$ of size $(n-1) * (n/2)$ which is passed from Algorithm 5 after filling valid entries. Inside Algorithm 8, entries of array $a[][]$ is tested for balanced tournament according to criteria discussed in Section 2.

Algorithm 8 Balanced Tournament Determination

```
1: procedure BTD(a, n)
2: begin
3: total := 0
4: for m := 2 to n − 1 do
5:     begin
6:     s := 0
7:     for j := 1 to n/2 do
8:         begin
9:         c1 := 0, c2 := 0
10:        for i := 1 to m − 1 do
11:            begin
12:            if (a[i][j].u = a[m][j].u)OR(a[i][j].v = a[m][j].u) then
13:                begin
14:                c1 := c1 + 1
15:                end
16:            if (a[i][j].u = a[m][j].v)OR(a[i][j].v = a[m][j].v) then
17:                begin
18:                c2 := c2 + 1
19:                end
20:            end
21:        if (c1 < 2)AND(c2 < 2) then
22:            begin
23:            s := s + 1
24:            end
25:        else
26:            begin
27:            return
28:            end
29:        end
30:    if s = (n/2) then
31:        begin
32:        total := total + 1
33:        end
34:    end
35: if total = n − 2 then
36:    begin
37:    File-Write(a, fp)
38:    HTD(a, n)
39:    end
40: end
```

After execution of this algorithm, we find that balanced tournament does not exist for value $n = 4$. But for value $n = 6$ balanced tournament exists. One balanced tournament, for $n = 6$ is shown in Table 4.8. Once balanced tournament determination is done, balanced tournament array $a[][]$ is passed to Hamilton path

determination i.e Algorithm 9, see line 38 in Algorithm 8.

Table 4.8: Balanced Tournament designs in different stadiums without any Hamilton paths, for $n = 6$

	Stadium 1	Stadium 2	Stadium 3
Day 1	(1, 2)	(3, 4)	(5, 6)
Day 2	(1, 3)	(2, 5)	(4, 6)
Day 3	(2, 6)	(3, 5)	(4, 1)
Day 4	(3, 6)	(4, 2)	(5, 1)
Day 5	(4, 5)	(1, 6)	(2, 3)

4.5 Backtracking for Hamilton Path based Tournament Designs

In this section we are going to discuss about Hamilton path based tournament determination. The definition of Hamilton path based tournament is described is section 1 and section 2. In brief, it must be balanced and set of games played in each stadium during $(n-1)$ days must form a Hamilton path. Algorithm 8, 9[27], and 10[27] covers the whole process of Hamilton path detection. Algorithm 8 i.e HTD(a, n) receives balanced tournament array $a[][]$ from Algorithm 7 i.e BTD(a, n). Inside Algorithm 8 i.e HTD(a, n) global 2-D array $G[][]$ maintains adjacency matrix formed by set of games played in a particular stadium during $(n-1)$ days. Further, Algorithm 8 i.e HTD(a, n) invokes backtrack Algorithm 9 i.e Hamilton(2, n) for each stadium, see line 18. Algorithm 9 i.e Hamilton(2, n) calls a criteria function Next-Value(i, n) i.e Algorithm 10 that uses adjacency matrix i.e global 2-D array $G[][]$ for detection of Hamilton path.

Table 4.9: Balanced Tournament designs having Hamilton paths in three different stadium 1, 2 and 4

	Combination No.	Stadium 1	Stadium 2	Stadium 3	Stadium 4
Day 1	326	(1, 5)	(2, 6)	(3, 7)	(4, 8)
Day 2	365	(1, 6)	(3, 4)	(2, 8)	(7, 5)
Day 3	790	(2, 4)	(5, 8)	(6, 7)	(3, 1)
Day 4	1020	(2, 7)	(3, 8)	(1, 4)	(5, 6)
Day 5	1393	(3, 6)	(4, 7)	(5, 2)	(1, 8)
Day 6	1667	(4, 5)	(1, 7)	(6, 8)	(3, 2)
Day 7	2516	(7, 8)	(1, 2)	(5, 3)	(6, 4)

Algorithm 9 Hamilton path based tournament determination

```
1: procedure HTD(a, n)
2: begin
3: sum := 0
4: for k := 1 to n/2 do
5:     begin
6:     for i := 1 to n − 1 do
7:         begin
8:         G[(a[i][k].u)][(a[i][k].v)] := 1
9:         G[(a[i][k].v)][(a[i][k].u)] := 1
10:        end
11:    for i = 1 to n do
12:        begin
13:        x[1] := i
14:        for j := 2 to n do
15:            begin
16:            x[j] := 0
17:            end
18:        count[k] :=Hamilton(2, n)
19:        if count[k] = 1 then
20:            begin
21:            break
22:            end
23:        end
24:    for i := 1 to n do
25:        begin
26:        for j := 1 to n do
27:            begin
28:            G[i][j] := 0
29:            end
30:        end
31:    sum := sum + count[k]
32:    end
33: if sum := n/2 then
34:    begin
35:    for i := 1 to n − 1 do
36:        begin
37:        for j := 1 to n/2 do
38:            begin
39:            Print(a[i][j].u, a[i][j].v)
40:            end
41:        end
42:    end
43: end
```

Algorithm 10 Hamilton path detection [27]

```
1: procedure Hamilton(i, n)
2: begin
3: while true do
4:     begin
5:     Next-Value(i, n)
6:     if x[i] = 0 then
7:         begin
8:         return false
9:         end
10:    if i = n then
11:        begin
12:        return true
13:        end
14:    else
15:        begin
16:        return Hamilton(i + 1, n)
17:        end
18:    end
19: end
```

In section 3.5, we have declared that no balanced tournament for value $n = 4$. In this section, we have got another important result for $n = 6$. When all balanced tournament results for $n = 6$ are passed for Hamilton path detection, we do not obtain any Hamilton path based tournament. See Table 4.8, for $n = 6$, an example of balanced tournament which is not Hamilton path based tournament. Now the most awaited result for value $n = 8$, when balanced results are passed for Hamilton path detection then we obtain two kind of interesting results. One is partial Hamilton path based tournament and another full Hamilton path based tournament. A partial Hamilton based tournament is shown in Table 4.9. It has three Hamilton paths in stadium 1, 2 and 4 respectively. The result of full Hamilton path based tournament has been given in Table 4.10. It contains total 4 Hamilton paths one in each stadium.

Table 4.10: Hamilton paths in each stadium, for $n = 8$

	Combination No.	Stadium 1	Stadium 2	Stadium 3	Stadium 4
Day 1	1	(1, 2)	(3, 4)	(5, 6)	(7, 8)
Day 2	152	(1, 3)	(5, 7)	(2, 4)	(6, 8)
Day 3	1098	(2, 8)	(3, 6)	(7, 1)	(5, 4)
Day 4	1518	(3, 7)	(6, 2)	(4, 8)	(5, 1)
Day 5	1761	(4, 6)	(1, 8)	(5, 3)	(7, 2)
Day 6	1863	(4, 7)	(5, 2)	(3, 8)	(1, 6)
Day 7	2194	(5, 8)	(4, 1)	(6, 7)	(3, 2)

Algorithm 11 Criteria Function for Hamilton path [27]

```
1: procedure Next-Value(i, n)
2: begin
3: while true do
4:     begin
5:     x[i] := (x[i] + 1)mod(n + 1)
6:     if x[i] = 0 then
7:         begin
8:         return
9:         end
10:    if G[x[i − 1]][x[i]] ≠ 0 then
11:        begin
12:        for j = 1 to i − 1 do
13:            begin
14:            if x[j] = x[i] then
15:                begin
16:                break
17:                end
18:            end
19:        if j = i then
20:            begin
21:            if i ≤ n then
22:                begin
23:                return
24:                end
25:            end
26:        end
27:    end
28: end
```

Chapter 5

Testing and Results

In last chapter, we have discussed algorithmic implementation of our problem. The whole work has been developed in C language. Further, in this chapter, different test cases and corresponding results would be discussed. In this problem, the test cases is mainly depending upon type and value of total numbers of teams n. It is covered in different sections.

5.1 For total number of teams, $n = 4$

There are different intermediate results obtained for value $n = 4$. It is shown in upcoming subsections of this section.

5.1.1 Days and Stadiums Distribution

At very first step, one need to check for days and stadium distribution according even value of $n = 4$. It works fine, see Figure 5.1 and 5.2. It determines days and stadium according round-robin tournament criteria.

5.1.2 Permutation Generation

After days and stadiums distribution, next intermediate step is permutation generation. It needs to generate $n!$ arrangements for input n. It works better, just see Figure 5.3.

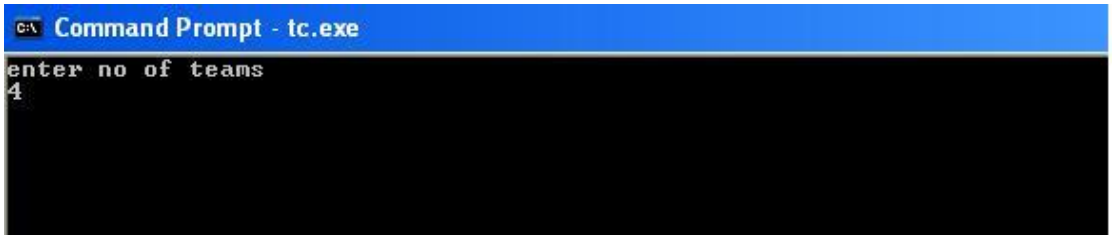

Figure 5.1: Input, $n = 4$

Figure 5.2: Even determination and distribution of days and stadiums, $n = 4$

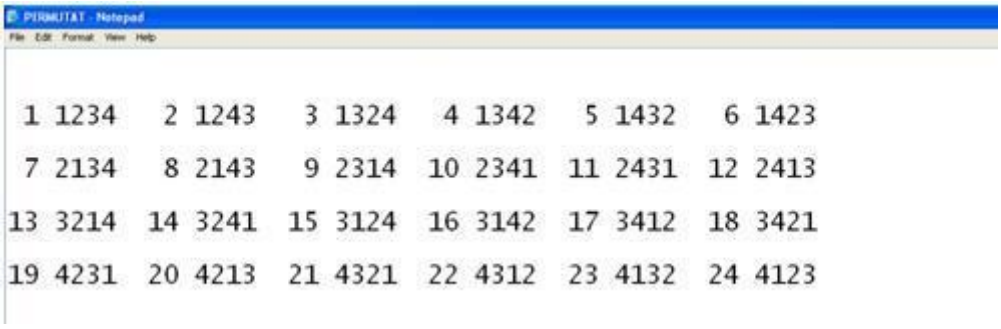

Figure 5.3: Total permutations for even value $n = 4$

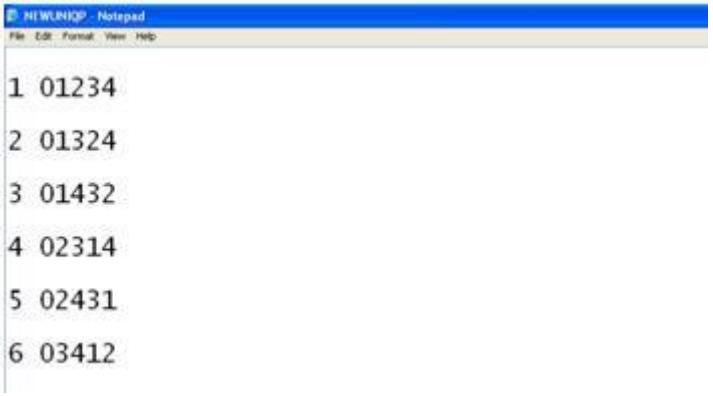

Figure 5.4: Unique permutations after removal of duplicates, $n = 4$

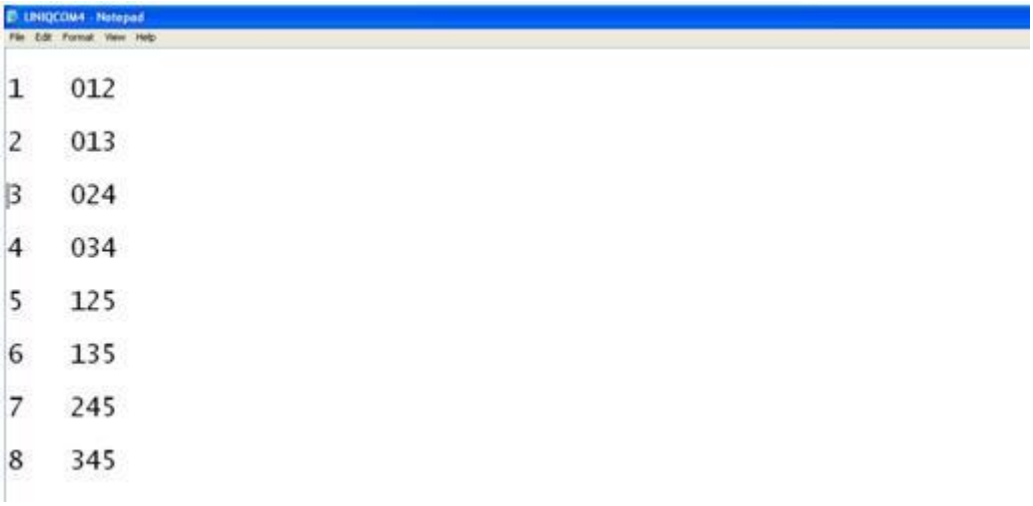

Figure 5.5: Unique Combinations for $n = 4$

5.1.3 Unique permutations after duplicates removal

Now, we perform next operation $firststepfilter$ on total permutation. It must generates total $t = 4!/(2! * 2!) = 6$, unique entries for input value, $n = 4$. It can be verified from output Figure 5.4.

5.1.4 Unique Combinations

The set of unique permutations are further used in generating set of unique combinations. At this stage, total $\binom{t}{n-1}$ combinations are generated. Each set is then filtered according balanced tournament criteria. For $n = 4$, there are total 8 unique permutation. It can be verified from output Figure 5.5.

5.1.5 Balanced and Hamilton Path Tournament Designs

For $n = 4$, we do not get any balanced tournament design. Therefore, if there is no balanced tournament designs, no possibility of Hamilton path tournament designs.

Figure 5.6: Input, $n = 6$

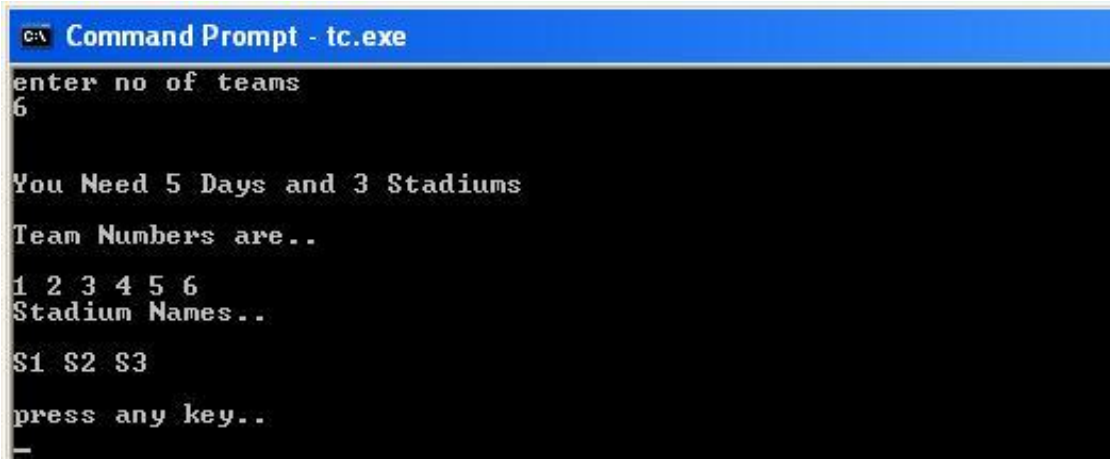

Figure 5.7: Even determination and distribution of days and stadiums, $n = 6$

5.2 For total number of teams, $n = 6$

There are different intermediate results obtained for value $n = 6$. It is shown in upcoming subsections of this section.

5.2.1 Days and Stadiums Distribution

In next test case, one need to check for days and stadium distribution according even value of $n = 6$. It works fine, see Figures 5.6 and 5.7. It determines days and stadium according round-robin tournament criteria.

5.2.2 Unique permutations after duplicates removal

After days and stadiums distribution, next intermediate step is permutation generation. It needs to generate $n!$ arrangements for input n. It works fine. Now, we perform next operation $firststepfilter$ on total permutation. It must generates total $t = 6!/(2! * 2! * 2!) = 90$, unique entries for input value, $n = 4$. It can be verified from output Figure 5.8.

5.2.3 Balanced Tournament Designs

For $n = 6$, we get many balanced tournament designs. See Figures 5.9 and 5.10, all satisfy balanced tournament criteria.

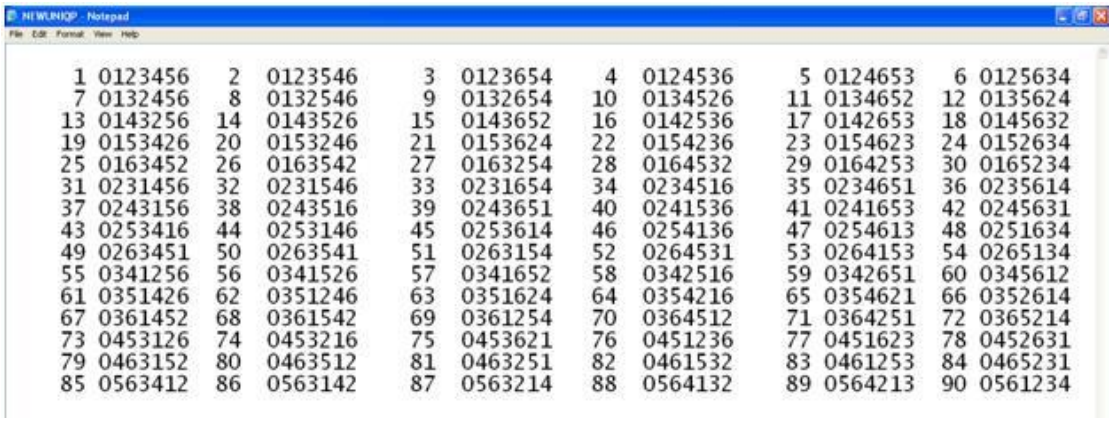

Figure 5.8: Unique permutations after removal of duplicates, $n = 6$

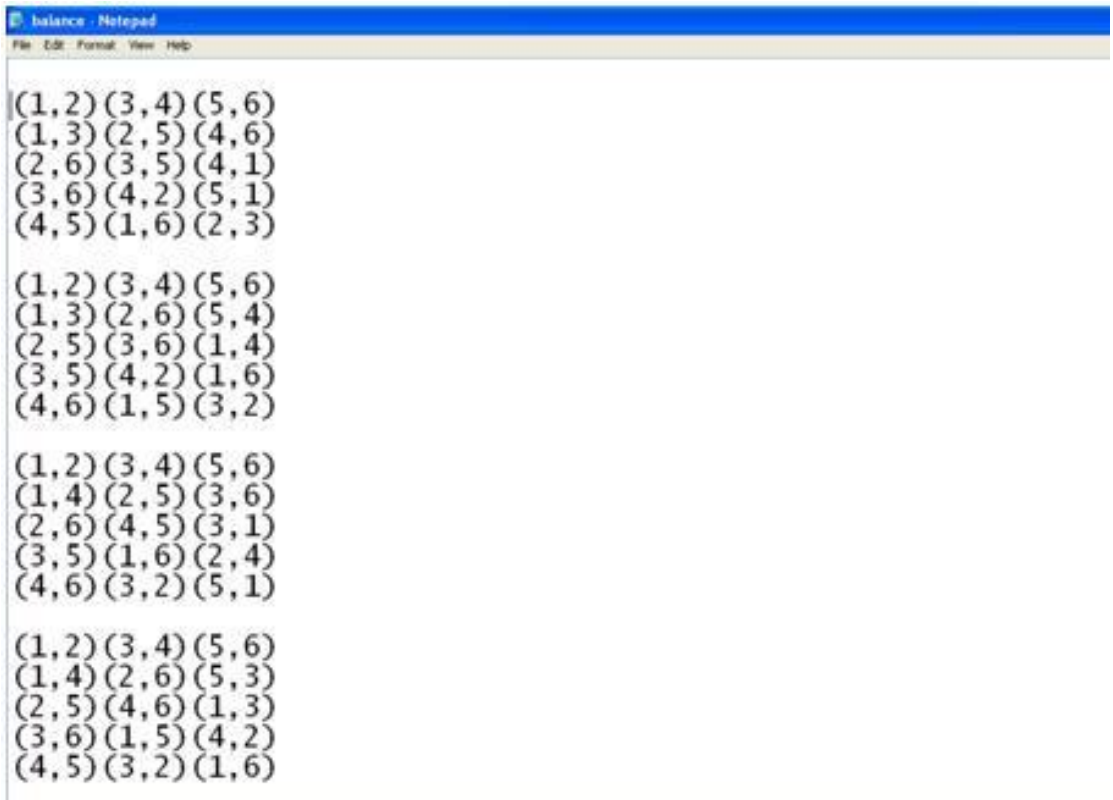

Figure 5.9: Balanced Tournament Designs-I, $n = 6$

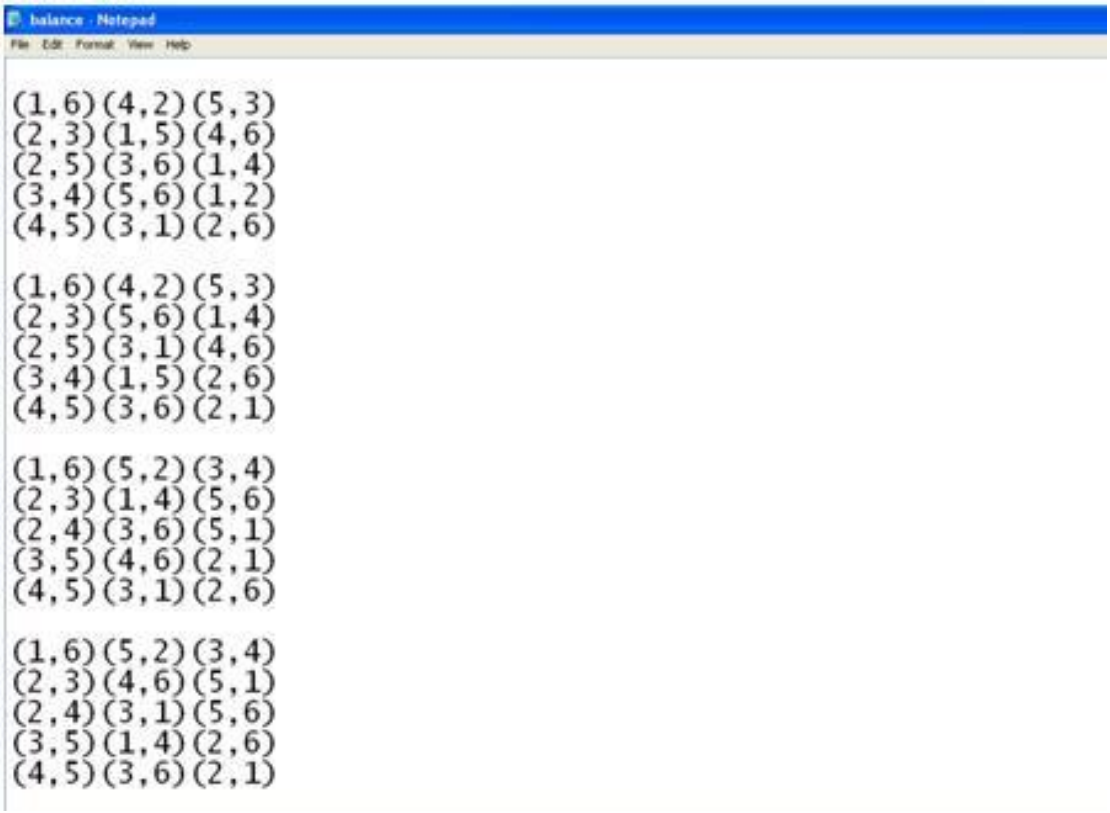

Figure 5.10: Balanced Tournament Designs-II, $n = 6$

5.2.4 Hamilton Path Tournament Designs

In case of $n = 6$, there is availability of balanced tournament designs. Therefore, it might be possible that Hamilton path tournament designs exist. But testing has been done for all unique combinations. We do not get any balanced tournament that satisfy Hamilton path tournament criteria.

5.3 For total number of teams, $n = 8$

The third test case is designed for value $n = 8$. There are different intermediate results obtained for value $n = 8$. It is shown in upcoming subsections of this section.

5.3.1 Days and Stadiums Distribution

In this test case, one need to check for days and stadium distribution according even value of $n = 8$. It works fine, see Figures 5.11 and 5.12. It determines days and stadium according round-robin tournament criteria.

Figure 5.11: Input, $n = 8$

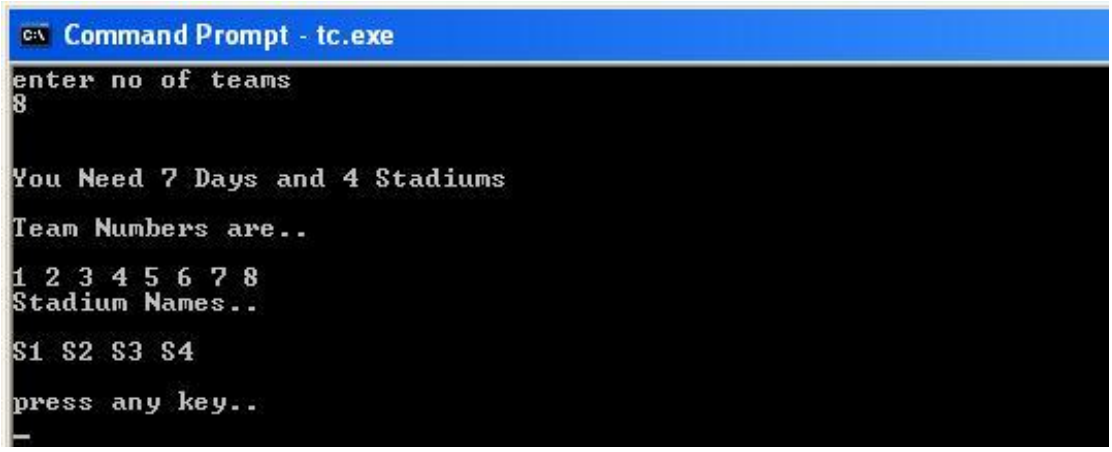

Figure 5.12: Even determination and distribution of days and stadiums, $n = 8$

5.3.2 Unique permutations after duplicates removal

After days and stadiums distribution, next intermediate step is permutation generation. It needs to generate $n!$ arrangements for input n. Now, we perform next operation $firststepfilter$ on total permutation. It must generates total $t = 8!/(2! * 2! * 2! * 2!) = 2520$, unique entries for input value, $n = 4$. It can be verified from output Figures 5.13 and 5.14. It is total 2520 in number. The first figure covers first 138 entries and second covers entries from 2497 to 2520.

5.3.3 Balanced Tournament Designs

For $n = 8$, we get many balanced tournament designs. One of such results is shown in Figure 5.15. It satisfies balanced tournament criteria.

5.3.4 Hamilton Path Tournament Designs

for value $n = 8$, when balanced results are passed for Hamilton path detection then we obtain two kind of interesting results. One is partial Hamilton path based tournament and another full Hamilton path based tournament. A partial Hamilton based tournament is shown in Figure 5.15. It has three Hamilton paths in stadium 1, 2 and 4 respectively. The result of full Hamilton path based tournament has been shown in Figure 5.16. It contains total 4 Hamilton paths one in each stadium.

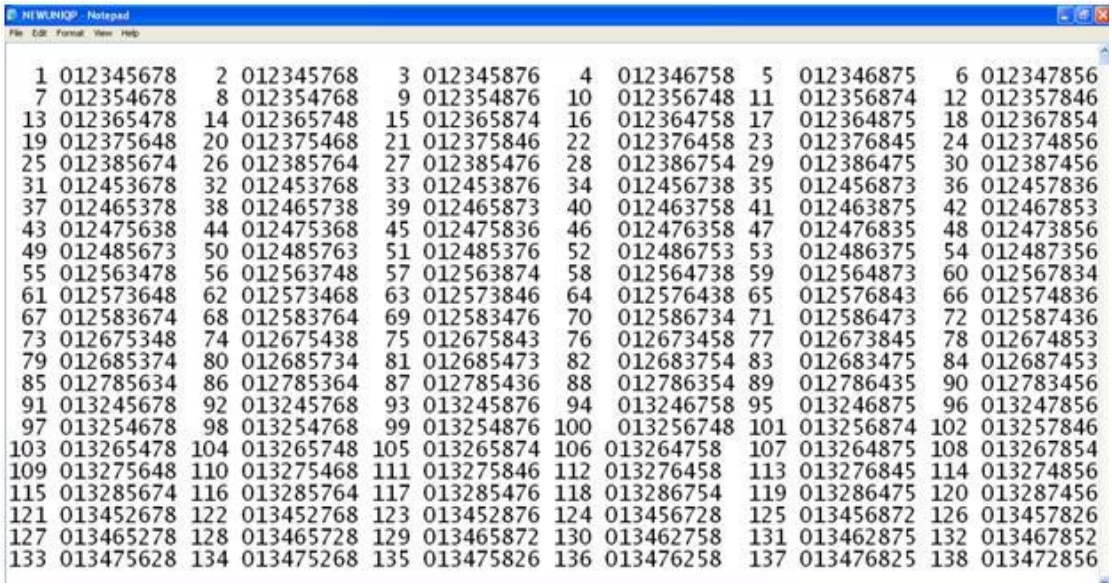

Figure 5.13: Unique permutations after removal of duplicates-I, $n = 8$

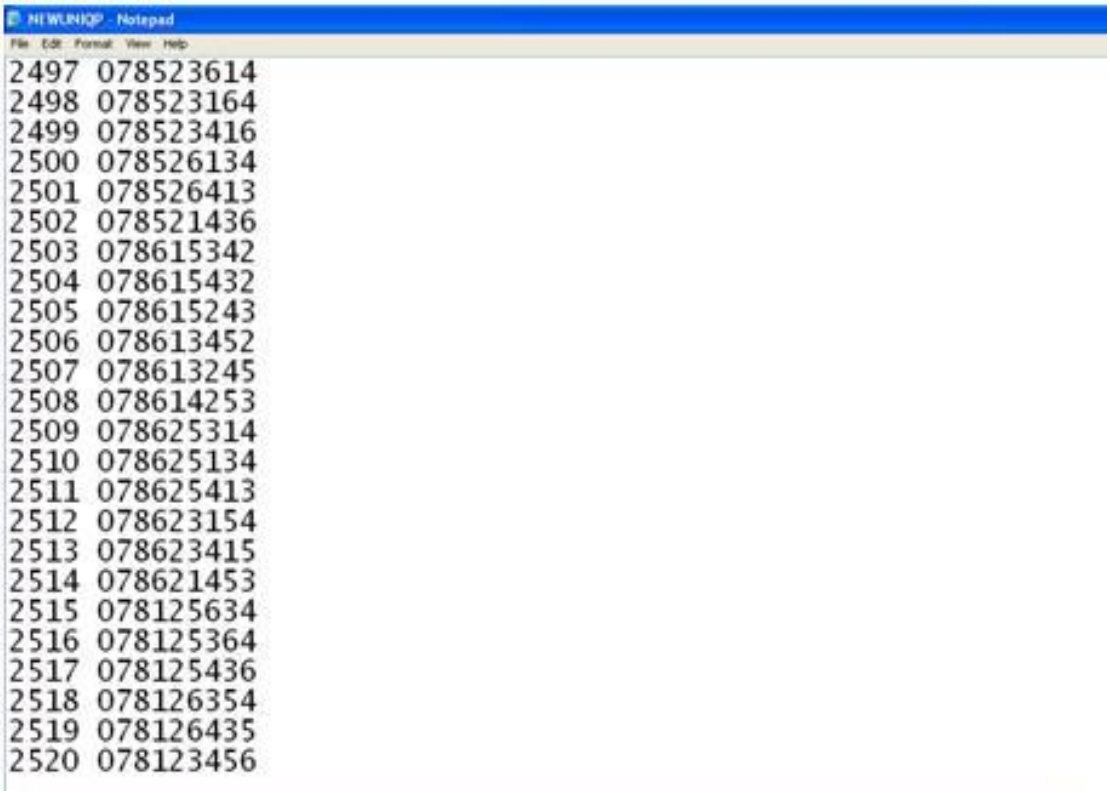

Figure 5.14: Unique permutations after removal of duplicates-I, $n = 8$

balance - Notepad

File Edit Format View Help

	Combination No.	Stadium 1	Stadium 2	Stadium 3	Stadium 4
Day 1	326	(1, 5)	(2, 6)	(3, 7)	(4, 8)
Day 2	365	(1, 6)	(3, 4)	(2, 8)	(7, 5)
Day 3	790	(2, 4)	(5, 8)	(6, 7)	(3, 1)
Day 4	1020	(2, 7)	(3, 8)	(1, 4)	(5, 6)
Day 5	1393	(3, 6)	(4, 7)	(5, 2)	(1, 8)
Day 6	1667	(4, 5)	(1, 7)	(6, 8)	(3, 2)
Day 7	2516	(7, 8)	(1, 2)	(5, 3)	(6, 4)

Figure 5.15: Balanced Tournament designs having Hamilton paths in three different stadium 1, 2 and 4

hmpath8 - Notepad

File Edit Format View Help

	Combination No.	Stadium 1	Stadium 2	Stadium 3	Stadium 4
Day 1	1	(1, 2)	(3, 4)	(5, 6)	(7, 8)
Day 2	152	(1, 3)	(5, 7)	(2, 4)	(6, 8)
Day 3	1098	(2, 8)	(3, 6)	(7, 1)	(5, 4)
Day 4	1518	(3, 7)	(6, 2)	(4, 8)	(5, 1)
Day 5	1761	(4, 6)	(1, 8)	(5, 3)	(7, 2)
Day 6	1863	(4, 7)	(5, 2)	(3, 8)	(1, 6)
Day 7	2194	(5, 8)	(4, 1)	(6, 7)	(3, 2)

Figure 5.16: Hamilton path tournament design, $n = 8$

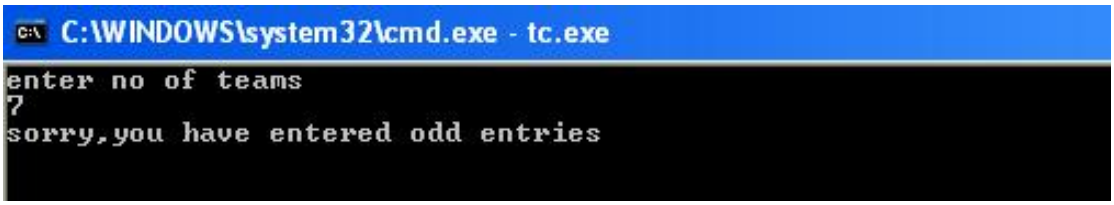

Figure 5.17: Determination of Odd entries

5.4 For odd value of total number of teams, like $n = 7$

The last test case is for determination of odd entries because the whole work of this thesis is based on even value of n. We get a proper message regarding this input, see Figure 5.17.

Chapter 6

Conclusion and Future Work

We have shown the presence of Hamilton path tournament designs for $n = 2^p \geq 8(p \geq 3)$ teams. In this problem-solution the complete graph is base. Many other graph scenarios may be covered on application basis with this approach. It can be extended as simulator or commercial product for balanced tournament generator in National and International games with efficient allocation of stadiums. It can be useful as research tool in areas where detection of Hamilton paths is done as a step process. The different ways Hamilton paths formation and 2-factors are interesting topics for future research.

The practical applications of Graph algorithms are widely in our daily life. They are used in market research, economy, aeronautics, physics, biology (for analyzing DNA), mathematics and other areas. Also, it is applicable in designing computer games, maps, drawing a circuit with a plotter in a fastest possible way, or with a minimum cost. It may be used to determine the cheapest path for garbage collection, street cleaning, or snow removal. Also applied in routing robots, packets in networking and others. As we all know to find a Hamilton path in a graph, is a NP-complete problem. Therefore, presented algorithm can used as a methodology in various approximation algorithms for generating approximate solutions of different NP problems. It can also applicable in class of problems that occur in cluster environments and require good communication efciency. That means, in cluster computing for finding Hamiltonian paths in tournaments on clusters in communication-efficient way.

References

[1] Yoshiko T. Ikebe and Akihisa Tamura, *Construction of Hamilton Path Tournament Designs*, Graphs and Combinatorics, Springer 2011, vol. 27, pp. 703–711, DOI 10.1007/s00373-010-0998-6, 2011.

[2] *Fifa world cup*, `http://www.fifa.com/worldcup/archive/southafrica2010/matches/index.html`, last visited on Jan, 2013.

[3] *UEFA*, `http://www.uefa.com/uefachampionsleague/season=2004`, last visited on Feb, 2013.

[4] *CONCACAF Gold Cup*, `http://www.goldcup.org/page/GoldCup/Schedule/0,,12802,00.html`,last visited on Feb, 2013.

[5] *World chess championship*, `http://www.fide.com/fide/handbook.html?id=20&view=category`, last visited on Jan, 2013.

[6] *Super rugby*, `http://www.supersport.com/rugby/super-rugby`, last visited on Jan, 2013.

[7] *Cricket World Cup*, `http://www.cricketworldcup.com/about`, last visited on April, 2013.

[8] *IPL*, `http://www.iplt20.com/results`, last visited on April, 2013.

[9] *Euroleague*, `http://www.euroleague.net/main/schedules/by-date?gamenumber=1&phasetypecode=RS`, last visited on Feb, 2013.

[10] *United footbaal league*, `http://www.ufl-football.com`, last visited on Feb, 2013.

[11] Jeffrey H. Dinitz., *Enumeration of Balanced Tournament Designs on 10 points*, Dept. of Mathematics and Statistics, University of Vermont, Burlington, VT 05405, pp. 1-12, `http://www.cs.cmu.edu/~mdinitz`, 2003.

[12] Narsingh Deo, *Coloring, Covering, and Partitioning*, Graph Theory with applications to engineering and computer science, PHI publication, ISBN-81-203-0145-5, First Edition, Ch. 8, pp. 165-190, 2004.

[13] Eppstein, David, *Regular labelings and geometric structures*, Proc. 22nd Canadian Conference on Computational Geometry (CCCG 2010), University of Manitoba, arXiv:1007.0221, 2010.

[14] Perkovic and Reed, *Edge coloring regular graphs of high degree*, Discrete Mathematics, pp. 165-166:567–578, doi:10.1016/S0012-365X(96)00202-6, 1997.

[15] Diestel and Reinhard, *Matching, covering and packing*, Graph Theory (3rd Electronic edition), Springer 2005, ISBN 3-540-26182-6, ch. 2, 2005.

[16] A. Prasant, Gopal Kishore, Kothapalli and V. Ch. Venkaiah, *Various One-Factorizations of Complete Graphs*, Center for Security, Theory, and Algorithmic Research, IIIT, Gachibowli, Hyderabad, India, p. 3, `http://www.people.csail.mit.edu/prasant/factorizations.pdf`, April 2007.

[17] Harary and Frank, *Factorization*, Graph Theory, Addison-Wesley, ISBN 0-201-02787-9, Chapter 9: Theorem 9.1, p. 85, 1969.

[18] Horton and J.D., *Hamilton path tournament designs*, Ars Combinatoria, vol. 27, pp. 69–74, 1989.

[19] Narsingh Deo, *Tree, and Fundamental Circuits*, Graph Theory with applications to engineering and computer science, PHI publication, ISBN-81-203-0145-5, First Edition, Ch. 3, pp. 39-64, 2004.

[20] Chetwynd, A. G. Hilton, A. J. W., *Regular graphs of high degree are 1-factorizable*, Proceedings of the London Mathematical Society 50 (2), pp. 193-206, doi:10.1112/plms/s3-50.2.193, 1985.

[21] J. H. Dinitz, P. Dukes and D. R. Stinson, *Sequentially perfect and uniform one-factorizations of the complete graph*, The Electronic journal of combinatorics, vol. 12, 2005.

[22] Gelling, E.N., Odeh, and R.E., *1-factorizations of the complete graph and the relationship to round robin tournaments*, Proc. Third Manitoba Conference on Numerical Math., Winnipeg 213–221, 1973.

[23] Schellenberg, P.J., van Rees, G.H.J., Vanstone, and S.A., *The existence of balanced tournament design*, Ars Combinatoria, vol. 3, pp. 303–317, 1977.

[24] Chengmin Wang, Jie Yan, *The Existence of Near Generalized Balanced Tournament Designs*, The Electronic journal of combinatorics, vol. 19, 2012.

[25] Sahni et al., *Recursive Permutation*, Computer Algorithms, Galgotia Publications, 2nd ed., ch. 1, p. 13, 2009.

[26] Martin Broadhurst, *combination.c - the next combination algorithm*, Copyright (C) 2010, `http://www.martinbroadhurst.com/source/combination.c.html`.

[27] Sahni et al., *Backtracking*, Computer Algorithms, Galgotia Publications, 2nd ed., ch. 7, pp. 347–350, 2009.

Publications

1. Desh Deepak Pathak and Ravinder Kumar, *Graph Factorization and Hamilton Path Based Balanced Tournament Designs* has been communicated in June 2013 with journal *Graphs and Combinatorics* [Springer].