A Tiling Approach to

Producing

High Quality Solutions

To

Real World Timetabling Problems

by

Douglas L. Moody

A Dissertation submitted to the Graduate Faculty in Computer Science in partial fulfillment of the requirements for the degree of Doctor of Philosophy, The City University of New York

2011

This manuscript has been read and accepted for the Graduate Faculty in Computer Science in satisfaction of the requirement for the degree of Doctor of Philosophy.

March 15, 2011
Date

Professor Amotz Bar-Noy
Chair of Examining Committee
Brooklyn College
The City University of New York

March 15, 2011
Date

Professor Theodore Brown
Executive Officer
The Graduate Center
The City University of New York

March 15, 2011
Date

Professor Graham Kendall
Professor in Computer Science
School of Computer Science and Information Technology
University of Nottingham
Nottingham NG8 1BB, UK

March 15, 2011
Date

Professor Xiangdong Li
Associate Professor
New York City College Of Technology
The City University of New York

Supervisory Committee

The City University of New York

Abstract

A Tiling Approach to Producing High Quality Solutions to Real World Timetabling Problems

By

Douglas L. Moody

Advisor: Professor Amotz Bar-Noy

The problem of constructing timetables in a fast and efficient manner is faced by a wide variety of scheduling practitioners. These timetables may specify the weekly calendar of university courses for a semester or contain the dates and times of games for a sports league. Current areas of research in timetabling have focused on problem instances that are small and often not representative of the real world challenges. Hence, the solution approaches proposed often do not solve the needs of the scheduling practitioner.

This thesis describes a robust and efficient technique using a new approach, referred to as "tiling, "to provide high quality solutions for a variety of timetabling problems. The tiling method is an efficient and flexible method capable of managing the size and complexity of real world timetables. The tiles in this method represent conjectures about the relationship among the events to be scheduled within a timetabling problem. A constructive approach is used to create an initial solution in phase one of the tiling method. A second phase of local search is used to tune the solution. We demonstrate the

flexibility of the approach through its application to large and small problems in various timetabling areas.

By applying our method to problems in previous studies, we are able to demonstrate how our tiles are found within the best known solutions to date, and are comparable in quality with respect to the objective function. Yet the tiling approach only uses a small fraction of the computing resources used in the best approaches.

Using our tiling approach, we directly address real world timetables to demonstrate its adaptability to real world scenarios. For example, the tiling approach has successfully been applied to sports leagues containing over five hundred teams for a youth sports league. In addition to the number of teams involved, this scheduling problem has other layers of complexity not previously addressed in other studies. Importantly, these leagues are more numerous than the professional sports, which have been the subject of most research to date. We introduce and define a new sports scheduling problem to encourage the further study of more common real world timetabling challenges.

ACKNOWLEDGEMENTS

Several people were very helpful in providing guidance and encouragement in my pursuit of the Ph.D. Dr. Amotz Bar-Noy provided excellent teaching from my first class in the Graduate Center through the entire dissertation process. I greatly appreciate his effort as my mentor. Dr. Graham Kendall was instrumental in introducing me to the timetabling discipline and leading me in the proper research directions. His help and guidance was invaluable to my work. Dr. Theodore Brown, Executive Officer of the Ph/.D. program at the Graduate Center of CUNY has been a very supportive member of the committee. Dr. Brown encouraged me from the beginning to continue in the program and overcome various obstacles. Assistants Lina Garcia of the CUNY Graduate Center and Deborah Pitchfork of Nottingham helped greatly in maintaining the communication flow. Anne Marinovic supplied great help in proofreading, often at a moment's notice. Fellow students Allen Harper and Anne Thompson-Smith were also very helpful and provided support during some key times.

However, this effort would not have been possible without my loving wife, Irene, whose support and encouragement spurred me to continue the effort. Her handling of family commitments gave me the necessary time to work on the project. My children, Tara, Brendan, Shauna and Bridget were also supportive and a source of inspiration.

TABLE OF CONTENTS

CHAPTER 1- INTRODUCTION 1

1.1 Overview 1

1.2 Motivation 2

1.3 Scope and Objectives 4

1.4 Summary 5

1.5 Outline 6

CHAPTER 2 – TIMETABLING SOLUTION GENERATION TECHNIQUES 8

2.1 Introduction 8

2.2 Solution Approach Phases 10

2.3 Local Search Techniques 12

2.3.1 Greedy Local Search 12

2.3.2 Great Deluge Algorithm 14

2.3.3 GRASP Algorithm 15

2.3.4 Simulated Annealing 17

2.3.5 Tabu Search 19

2.4 Integer Programming 20

2.4.1 Integer Programming Related Research 21

2.4.2 Integer Programming Example 21

2.4.3 Integer Programming and Constraint Processing 22

2.5 Constraint PROGRAMMING 23

2.5.1 Constraint Satisfaction Example 25

2.5.2 Constraint Programming Tools 27

2.5.2.1 Variable Selection and Assignments 27

2.5.2.2 Consistency 29

2.5.2.3 Backtracking 30

2.6 Summary 31

CHAPTER 3 - TILING APPROACH 32

3.1 Introduction 32

3.2 Phase I of Tiling 33

3.2.1 Tile Creation 34

3.2.2 Tile Costing 35

3.2.3 Tile Placement 36

3.2.4 Consistency Checking 37

3.2.5 Backtracking and Phase I completion 38

3.3 Phase II – Local Search 38

3.4 Tiling Algorithm 39

CHAPTER 4 - CONSTRUCTING INITIAL NEIGHBORHOODS TO IDENTIFY CRITICAL CONSTRAINTS 43

4.1 Introduction 43

4.2 Problem Definition and Initial solution Generation 45

4.3 INitial Solution Analysis and Local Search Phase 46

4.4 Results 50

4.5 Conclusion 50

CHAPTER 5 - Construction of Initial Solutions for a Course Scheduling Problem Using Tiling 52

5.1 Introduction 52
5.2 Problem Definition 53
5.3 Our Approach 55
5.4 Tiling Concept 57
5.5 Pre-processing Steps 61
5.6 Tile Sets and Schedule Completion 62
5.7 Tuning the Schedule 64
5.8 The Complete Algorithm 66
5.9 Results 70
5.10 Conclusions 74
CHAPTER 6 - TTP SPORTS SCHEDULING PROBELM 77
6.1 Introduction 77
6.2 Related Work 79
6.3 Proposed Approach 81
6.3.1 Tile Creation 81
6.3.2 Tile Breaking Cost 85
6.3.3 Tile Placement 93
6.3.4 Local Search 94
6.4. Instances and Current Results 95
6.5. Our Results 99
6.6 Scalability and Flexibility 102
6.7 Conclusion 104
CHAPTER 7 – YOUTH SPORTS LEAGUES 107

7.1 Introduction 107
7.2. Background 110
7.2.1 League Formation 110
7.2.2 Instances of the YSL in the Real World 112
7.3 Problem Definition 112
7.4 Example YSL Instance 119
7.5 Related Work 123
7.6 . Proposed Approach 124
7.6.1 Tiling Phase 125
7.6.2 Local Search Phase 127
7.7 Results 129
7.8 Conclusions 131
CHAPTER 8 – CONCLUSIONS 133
8.1. Summary 133
8.2. Future Work 134
APPENDIX 137
BIBLIOGRAPHY 177

LIST OF FIGURES

Figure 2.1 – Course Scheduling Timetabling for ENG101,HIST102,CHEM112. 9

Figure 2.2 – Sports League Timetabling – Team Abbreviation by Week. 9

Figure 2.3 – Greedy Local Search Algorithm Pseudo-code. 14

Figure 2.4 – Great Deluge Algorithm Pseudo-code..... 15

Figure 2.5 – GRASP Algorithm Pseudo-code. 17

Figure 2.6 – Simulated Annealing Algorithm Pseudo-code. 18

Figure 2.7 – Tabu Search Algorithm Pseudo-code. 20

Figure 3.1 - Complete Algorithm for Tiling 41

Figure 4.1 Results From Instance 1 49

Figure 4.2 Results From Instance 8 49

Figure 5.1- Tiles slotted in periods 5 and 6. 58

Figure 5.2 - Tile placement within the schedule. 59

Figure 5.3- Moving Last Period Events into the Schedule. 65

Figure 5.4 - Algorithm Variables. 67

Figure 5.5 - Algorithm Functions. 67

Figure 5.6 - Algorithm for the Main Routine 68

Figure 5.7 - Algorithm for the Greedy Schedule Routine. 69

Figure 5.8 - Algorithm for the Exchange Event Routines 69

Figure 5.9 - Algorithm for the Process Initialization of Arrays Routine. 70

Figure 5.10- Algorithm for the Make TileSet Routine. 70

Figure 5.11 – Results, best results **bolded**. 72

Figure 5.12 - Key Indicators per instance 73

Figure. 6.1 Tile Placement for Teams 1 and 2 (shaded cells indicate away games). 81

Figure. 6.2 Minimum Spanning Tree (MST) for PIT in NL6 and the accompanying distance matrix. 83

Figure. 6.3 Creation of Tiles through collapsing of the tree. 85

Figure. 6.4. Example Tile Costing Diagram 86

Figure 7.1– Team calendar view perspective. 115

Figure 7.2– Team season view 115

Figure 7.3– Daily venue schedule. 115

Figure 7.1 – Tiling by Round Algorithm 129

LIST OF TABLES

Table 2.1 Sample Proctor / Exam Relationship. 22

Table 2.2 Sample Proctor / Exam Solution. 22

Table 2.3 Cost Data for Sports Scheduling Example 26

Table 3.1 Sample Tiles. 34

Table 3.2. Tiling Algorithm Variables 39

Table 3.3 Tiling Algorithm Functions 40

Table 6.1 Examples of tile value calculations. 88

Table 6.2. NFL 16 team instance comparison. 90

Table 6.3. NFL 22 team instance comparison. 91

Table 6.4. NFL 26 team instance comparison. 92

Table 6.5. Best NL instance results**.** 96

Table 6.6. Best NFL instance results NFL16 through NFL24**.** 97

Table 6.7. Best NFL instance results NFL26 through NFL32**.** 97

Table 6.8. Best TTP Constant Distance Results**.** 98

Table 6.9 Results Comparison (best solutions shown in **bold**). 99

Table 6.10**:** Time Comparison in seconds. 100

Table 6.11: New TTP instance definition. 103

Table 6.12**:** New TTP instance results. 104

Table 7.4 – Sample problem input parameters and data. 121

Table 7.5 – Tiles (Rounds) for the example league divisions 122

Table 7.6 – Sample Problem Scheduling Grid 123

Table 7.7 – Unbalanced Schedule in terms of tournaments 126

Table 7.8 – Venue related soft constraint violation results of WPCYO. 130
Table 7.9 – Venue related soft constraint violation results of WPCYO. 131
Table A1 - Tile Comparison for 16 team instance 140
Table A2 - Tile Comparison for 22 team instance 145
Table A3 - Tile Comparison for 26 team instance 149
Table A4 - Table Distance Matrix for DIV 1 Instance 150
Table A5 - Match-Ups for DIV 1 Instance 150
Table A6 - Tiles for DIV 1 Instance 150
Table A7 - Solution for DIV 1 Instance (- indicates home game) – Part I 151
Table A8 - Solution for DIV 1 Instance (- indicates home game) – Part II 151
Table A9 - Table Distance Matrix for DIV 2 Instance 151
Table A10 - Match-Ups for DIV 2 Instance 152
Table A11 - Tiles for DIV 2 Instance Part I 152
Table A12 - Tiles for DIV 2 Instance Part II 152
Table A13 - Solution for DIV 2 Instance (- indicates home game) – Part I 153
Table A14 - Solution for DIV 2 Instance (- indicates home game) – Part II 153
Table A15 - Table Distance Matrix for DIV 4 Instance – PART I 154
Table A16 - Table Distance Matrix for DIV 4 Instance – PART II 155
Table A17 - Match-Ups for DIV 4 Instance – Part I 156
Table A18 - Match-Ups for DIV 4 Instance – Part I 157
Table A19 - Tiles for DIV 4 Instance Part I 157
Table A20 - Tiles for DIV 4 Instance Part II 158
Table A21 - Tiles for DIV 4 Instance Part III 158

Table A22 - Tiles for DIV 4 Instance Part IV 158
Table A23 - Solution for DIV 4 Instance (- indicates home game) – Part I 159
Table A24 - Solution for DIV 4 Instance (- indicates home game) – Part II..... 160
Table A25 - Solution for DIV 4 Instance (- indicates home game) – Part III 161
Table A26 - Table Distance Matrix for Actual MLB Schedule Instance – PART I..... 162
Table A27 - Table Distance Matrix for Actual MLB Schedule Instance – PART II 163
Table A28 - Match-Ups for Actual MLB Instance – Part I 164
Table A29 - Match-Ups for Actual MLB Instance – Part II..... 166
Table A30 - Tiles for Actual MLB Instance Part I 166
Table A31 - Tiles for Actual MLB Instance Part II..... 166
Table A32 - Tiles for Actual MLB Instance Part III 167
Table A33 - Tiles for Actual MLB Instance Part IV 167
Table A34 - Tiles for Actual MLB Instance Part V..... 168
Table A35 - Tiles for Actual MLB Instance Part VI 168
Table A36 - Tiling Solution for Actual MLB Instance (- indicates home game) – Part I
..... 169
Table A37 - Tiling Solution for Actual MLB Instance (- indicates home game) – Part II
..... 170
Table A38 - Tiling Solution for Actual MLB Instance (- indicates home game) – Part III
..... 171
Table A39 - Tiling Solution for Actual MLB Instance (- indicates home game) – Part IV
..... 172
Table A40 - Actual MLB Schedule (- indicates home game) – Part I..... 173

Table A41 - Actual MLB Schedule (- indicates home game) – Part II 174

Table A42 - Actual MLB Schedule (- indicates home game) – Part III 175

Table A43 - Actual MLB Schedule (- indicates home game) – Part IV 176

CHAPTER 1- INTRODUCTION

1.1 OVERVIEW

The field of timetabling has drawn considerable attention over the last decade due to the recent advances in both hardware and software technology. Applications of timetabling include constructing university schedules of course offerings, organizing athletic events and creating rosters of on duty medical personnel. An introduction to timetabling is presented by de Werra [32]. The increased power of the personal computer has enabled timetablers to have the processing power previously only available in the mainframe environment. Software tools, including constraint processing packages, have been developed and can be used on the personal computer platform. This new personal computer environment can now provide support for sophisticated and resource-intensive approaches to solving real world timetabling problems.

One of the most common real world problem studied, and directly experienced by academia, is that of course scheduling. The goal of this problem is to create a timetable that represents the schedule of events, or courses, over a portion of the academic year. This timetable consists of days and time periods during the week referred to as slots. Generally, this requires the assignment of resources, such as rooms and professors, to the set of offered courses. The assignment task allocates resources to a course at a particular slot in a timetable. In addition to course scheduling, a related problem of examination

scheduling is also widely studied and has similar goals and constraints as the course scheduling problem. This problem requires students to take a set of examinations over a period of time, within the timetable.

Another highly visible timetabling problem is sports scheduling. Professional sports are followed by billions of people worldwide, and hence their schedule affects a large number of people. The schedule of a professional sports league can impact both the travel plans and the interest level of the fan base. A more widespread application of sports scheduling is the scheduling of amateur sports, particularly youth sports leagues. Organized amateur sports are played directly by millions of people and this presents a vast and novel opportunity to use sophisticated timetabling approaches. Sports scheduling problems, like course scheduling, have a timetable which holds the schedule of the events (or games), within the league. Games must be assigned teams and a venue for the game. Similar to class scheduling, an assignment designates teams to slots within the timetable.

1.2 MOTIVATION

The current state of research in course and sports timetabling is highly focused on local search techniques typically employing constraint processing. Local search techniques, including tabu search and simulated annealing, are widely used in the solution approaches to these academic and real world problems. These approaches have been successful at providing high quality solutions to some problems. However, often these solutions have tremendous costs in terms of computer resources. The resource intensive nature of these approaches limits their applicability. Due to their instance size limitation,

they consequently cannot handle real world scenarios containing a large number of events. Also, these approaches may not provide a timely solution to support the iterative nature of the timetabling process. Often the elapsed time needed for solution generation would be too long for a scheduling practitioner to revise the constraints and generate a new solution. These limitations have prevented the academic research in timetabling from becoming more widely applied to real world applications.

For example, in the academic based Travelling Tournament Problem (TTP) introduced by Trick [109], the largest instance of the problem defined is thirty-two teams. In addition, this problem has few constraints compared to real world scenarios. The best known approach by Van Hentenryck [115], uses a set of sixty servers to find the solution. This level of resource utilization would only be available to a small set of professional sports leagues. Other approaches [34] [6] also require long run times to produce their result. These run times inhibit the ability of the scheduler to revise the input constraints after solution examination, in order to include newly discovered constraints. In addition, these approaches make use of commercially expensive constraint programming software packages. Consequently, cost would hamper most organizations from adopting the approach.

This thesis looks to provide a timetabling approach that can be employed by individual users with limited computer resources and software tools. These individuals are not searching for the “optimal” solution or even the best approximate solution. Instead, these individuals need a facile tool that can meet all the constraints of their problem, and

produce a high quality solution within their processing environment. For this thesis, this environment is defined as a standard personal computer, without access to advanced constraint processing packages, requiring elapsed solution generation times of under one hour. This is a practical approach that can provide real world scheduling personnel with a workable tool.

1.3 SCOPE AND OBJECTIVES

This thesis introduces a new technique - "tiling" - that will help ameliorate some of the drawbacks of using known approaches, while still providing high quality solutions. Two major categories of problems, course scheduling and sports scheduling, are used to demonstrate the effectiveness of this new technique. In addition, we introduce a new sports scheduling problem that closely reflects real world amateur sports scheduling, rather than the scheduling of professional leagues, often studied in academia. This problem considers the schedule peculiarities, constraints, and size challenges faced by thousands of youth sports leagues.

The tiling technique represents an approach of incorporating conjectures of solution elements into the overall solution approach. Tiling provides a stronger initial solution from which to use other techniques, typically local search techniques, to find the best feasible solution. Our objectives include demonstrating how tiling can provide high quality initial solutions with minimal resources, as well as competitive feasible solutions. A last objective is to introduce new problems that truly capture the common challenges faced by schedulers, particularly in the sports arena.

1.4 SUMMARY

This thesis presents tiling as an approach to creating a high quality initial solution, and final solutions after a simple local search phase. The approach uses known information about the problem developed by a separate algorithm or through user experience. This information is embedded into the tiles. The tiles are places in the solution to the extent possible, based upon problem constraints. The tiles are then broken into their individual components and placed until the initial solution is complete. The local search phase is then performed on this initial solution.

The tiling technique is first shown to provide a high quality initial neighborhood for two course scheduling problems presented within a competition. The application of tiling leads to a much more effective initial solution by using less of the allowed computer run time to reach a higher quality initial solution than the winners of the competition. We demonstrate that a tiling approach can generate an initial solution in half the time and of nearly equal value to the best known approach.

The next major problem area addressed is sports scheduling. Here the focus is on the TTP problem. The tiling technique is used to find the find feasible solutions that are within 10-15% of the best known solutions using only a small fraction of the resources consumed by those other solutions. We use the tiling approach along with a common local search approach to find our solutions within minutes on one computer, versus the hours of computer network time used by our competitors.

We further suggest the potential of tiling by demonstrating that we can predict how our tiles can be found in these best known solutions. The tiles created and used by our solutions can be seen as the core of the best known solutions. We use a costing approach that can predict portions of the best known solutions.

The TTP has been instrumental in encouraging researchers to develop better techniques for timetablers. However, there are several characteristics of the TTP problem such as including instance size and the challenge of venue resource sharing, that have guided research in a direction not applicable for common real world problems. We introduce a new problem, the youth sports league problem, that encompasses the set of common sports scheduling constraints and instance size consideration, to move the field of timetabling from research to practical world application. We show how the tiling technique can provide a solution better than those solutions produced currently by timetablers.

1.5 OUTLINE

Chapter 2 of the thesis reviews the current approaches and techniques used to perform local search, a common phase in timetabling problem solution approaches, within course and sports scheduling. Chapter 3 looks closely at the tool used to implement local search and timetabling heuristics – constraint processing. Chapter 4 defines in detail the tiling approach introduced in this thesis. A description of each step in the approach is provided

and a general algorithm is documented. Comparisons are made to others local search approaches, and the relationship to general constraint processing is also explored.

Chapters 4 through 6 each covers a specific application of tiling to a timetabling problem that has previously been explored by other researchers. In these chapters, we show how the tiling approach can be used to produce results comparable to the best known results. Chapter 4 and Chapter 5 address course scheduling problem, while Chapter 6 concentrates on sports scheduling problems. Chapter 7 formally presents the previously unstudied youth sports scheduling problem that addresses the common scheduling challenges faced by thousands of sports leagues. This chapter also details how the tiling approach can work with one sample youth sports league. The last chapter summarizes our research in terms of providing high quality solutions and introducing real world scheduling problems.

CHAPTER 2 – TIMETABLING SOLUTION GENERATION TECHNIQUES

2.1 Introduction

The areas of course scheduling and sports scheduling share the common goal of event assignments to locations within a timetable. The event within course scheduling is a lecture. The timetable is the weekly schedule consisting of the days of the week, and hourly time periods. Each event is also assigned a location. A slot represents the time period associated with a location. Likewise, sports scheduling events are athletic contests or games between two opponents. The game is scheduled during a particular period, or week, of the season. This period, coupled with a location, is also referred to as a slot. The location of the game is an athletic facility available to one of the two teams competing in the game. The timetabling problems, for both these examples, are depicted in Figure 2.1 and Figure 2.2.

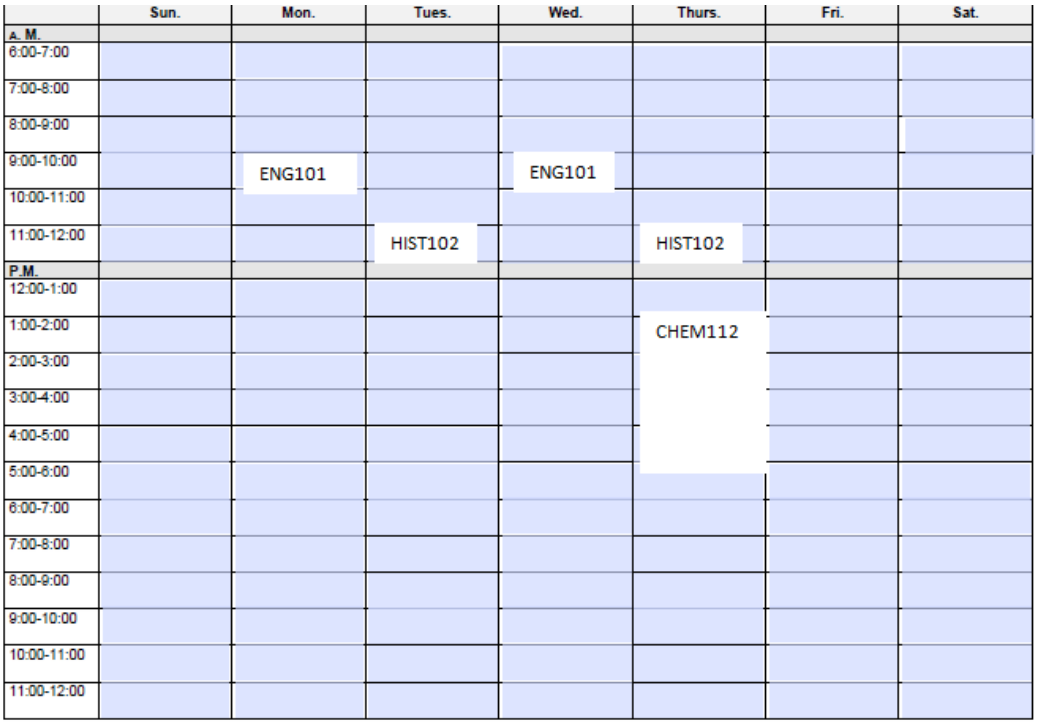


Figure 2.1 – Course Scheduling Timetabling for ENG101,HIST102,CHEM112.

TEAM	1	2	3	4	5	6	7	8
ARI	@STL	@ATL	OAK	@SD	NO	BYE	@SEA	TB
ATL	@PIT	ARI	@NO	SF	@CLE	@PHI	CIN	BYE
BAL	@NYJ	@CIN	CLE	@PIT	DEN	@NE	BUF	BYE
BUF	MIA	@GB	@NE	NYJ	JAC	BYE	@BAL	@KC
CAR	@NYG	TB	CIN	@NO	CHI	BYE	SF	@STL
CHI	DET	@DAL	GB	@NYG	@CAR	SEA	WSH	BYE
CIN	@NE	BAL	@CAR	@CLE	TB	BYE	@ATL	MIA
CLE	@TB	KC	@BAL	CIN	ATL	@PIT	@NO	BYE
DAL	@WSH	CHI	@HOU	BYE	TEN	@MIN	NYG	JAC
DEN	@JAC	SEA	IND	@TEN	@BAL	NYJ	OAK	@SF
DET	@CHI	PHI	@MIN	@GB	STL	@NYG	BYE	WSH

Figure 2.2 – Sports League Timetabling – Team Abbreviation by Week.

A solution to a timetabling problem is the schedule of events within slots that adhere to all the hard (or required) constraints of the problem. Constraints include ensuring no collisions of events in the location, having minimum or maximum events in a day, and satisfying any requirements regarding the use of an event's related resources. Timetabling problem solution quality are evaluated by many criteria. One measure is how well the timetable accommodates the soft (not required) constraints of the problems. Solutions with fewer soft constraint violations are noted to be higher quality. Another measure includes how a resource is used within the timetable. For example, the minimization of a team's travel distance is the goal in many sports problems.

The complexity of the timetabling problem is conjectured to NP-hard. For example, Bhattacharyya [13] proves the TTP problem is NP-hard without the constraint of consecutive home and away games, and Post & Woeginger [91] show aspects of the general sports tournament is NP-complete. Further, Cooper & Kingston [25] describe how a generic timetabling problem is NP-complete.

2.2 SOLUTION APPROACH PHASES

A variety of solution approaches are used to address timetabling problems. Rasmussen & Trick [94] and Kendall et al. [59] survey approaches to sports scheduling. Qu et al. [92] surveys approaches in examination scheduling. The majority of timetabling approaches are accomplished in two phases. The first phase creates an initial solution. This solution contains an assignment for all events and may or may not be a feasible solution, in which

all hard constraints are met. The second phase is a local search phase. This phase exchanges assignments of events to produce a feasible or higher quality solution. The initial solution for a timetabling problem can be generated through a heuristic method specific to the problem or by random assignments. Many approaches have little concern for the quality of the initial solution, and rely solely on the second phase of local search to provide a feasible and high quality solution. Anagnostopoulos et al. [6] and Kostuch [65] each are credited with the best timetabling in their area, yet they generate initial solutions that are virtually random.

The local search phase involves examining the search space, which encompasses the universe of feasible solutions. The initial solution from phase I places a timetabling approach at a certain position in the search space. A move is defined as a permutation of assignments which will alter the current solution into another feasible solution within the search space. This move involves the re-assignment of a set of decision variables resulting in the new solution. The move in course scheduling may be to move all classes from one day of the week to another day. For sports scheduling a move may be to switch the home and away team for a given game.

A neighborhood of a particular solution is defined as all the solutions in the search space that can be reached (or generated) via a particular type of move from this solution. For example, all solutions that can be created by switching one home and away team would be the neighborhood associated with that move. Local search techniques use moves to traverse the various solutions in the search space seeking the best solution.

Each move, and the subsequent evaluation of the solution reached, is referred to as an iteration. A maximum number of iterations is often defined to end the local search or proceed in a new search direction. In the next section, common local searches found in course and sports timetabling problems are discussed.

2.3 LOCAL SEARCH TECHNIQUES

All local search approaches have a defined set of moves, specific to the problem to reach a solution within the search space. A key objective of local search approaches is to explore the search space of high quality solutions, without spending time in areas offering little hope of a high quality solution. Each approach seeks to accomplish the divergent goals in different ways.

2.3.1 Greedy Local Search

The greedy local search is the most intuitive type of search. The approach analyzes a set of solutions in the neighborhood associated with a move. The “hill climbing” greedy algorithm selects a random move, and evaluates its effect the quality of the solution calculated by the objective function. If the move improves the solution, the move is made. The hill climbing approach terminates after a set number of iterations have been completed or when all moves do not improve the solution. Lim et al. [70] use hill climbing in conjunction with other local searches to find solutions for the Travelling Tournament Problem (TTP) The algorithm is used extensively in several scheduling problems. Dinitz & Stinson [37] use hill climbing to develop one factorizations, a key

element of tournament generation. Lim & Fu [69], Burke & Bykov [16], Ozcan et al. [89] offer hill climbing as part of the approach to examination timetabling. Ozcan [88] offers a scheduler tool which is based upon the approach. Course timetabling approaches involving hill climbing include Wang et al. [116] and Abuhamdah [3]. Nnuohiro & Maklin [86] employ hill-climbing within the decision-making capabilities of agents to address course timetabling, while Lewis [68] applies the approach to graph coloring.

The "steepest descent" approach of the greedy algorithm creates a set of all possible moves. The best move, in terms of minimizing the objective function, is selected from the set. The move is executed and another set is created based upon the new solution. This process continues until a solution is found where no moves to improve the current solution exist. Lim & Zhang [71] use this approach to achieve excellent results for the TTP.

In some cases, the greedy local search can be fast. It finds a local minimum quickly, and consequently only a small portion of the search space may be analyzed. The following pseudocode in figure 2.3 represents the hill climbing algorithm. The algorithm uses two procedures that are unique to the problem. The first is the *Generate_Initial_Solution* procedure which must generate a feasible solution as a starting point. The second procedure is the *Done* procedure which determines if the algorithm has run for a sufficient time given the problem definition and resources. The *Neighbor* procedure returns a solution that is a neighbor of S.

```
S = Generate_Initial_Solution
while (true)  {
        S` = Random(Neighbor(S))            // select a solution that is neighbor of S
                                            // and  removes solution from neighbor(S)
        if f(S`) < S then                   // is this neighbor above the water level
                S=S`                        // accept new solution if above water level
                                            // raise water level by r
        if Done()  then exit                // all neighbors have been tried
   }
```

Figure 2.3 – Greedy Local Search Algorithm Pseudo-code.

The next approaches seek to extend the searching task and reach more of the search space for one initial solution.

2.3.2 Great Deluge Algorithm

The great deluge algorithm seeks to "stay above water," where the water level is the initial solution value along with added improvements over time. It builds upon the concepts of the local search presented above and was introduced by Dueck [40]. Initially only improvements to the solution based on the objective function are accepted, driving the current value significantly above the initial solution or "water level". Later moves are accepted, regardless of their improvement to the current solutions, as long as the new value is above the current water level. The water level rises as more iterations are completed. This rise makes it more difficult to accept moves that reduce the current solution value, while staying above the water level. The only parameter necessary for the algorithm is the rate of the water rise from the initial solution. This rate, while problem specific, is easier to formulate than the cooling formula for simulated annealing, discussed in section 2.3.4.

This algorithm has been used within timetabling problems as a direct approach or in combination with other local search algorithms. Burke & Bykov [15] use this algorithm as their primary approach for a course scheduling competition. Extensions of the algorithm are used by McMullan [79] and Al-Milli [5] in course timetabling and McCollum [78] within examination timetabling.

The great deluge algorithm can applied to improve a solution *S*, whose objective function is f(*S*) in the following way. We assume that are trying to maximize f(*S*). We will define *r* as the rise in the water level W, or tolerance level, of the algorithm, and set it to f(*S*). The function *Neighbor***()** returns non-repetitive neighbor of S. The algorithm's pseudo-code is presented in figure 2.4.

```
W = Generate_Initial_Solution
S = W
while (f(S) ≥ W)  {
        S` = Random(Neighbor(S) )          // select a solution that is neighbor of S
                                           // and  removes solution from neighbor(S)
        if f(S`) > W then                  // is this neighbor above the water level
                S=S`                       // accept new solution if above water level
                W = W + r                  // raise water level by r
        if Neighbor(S) = null  then exit   // all neighbors have been tried
   }
```

Figure 2.4 – Great Deluge Algorithm Pseudo-code.

The solution *S* would be the best found solution for the algorithm.

2.3.3 GRASP Algorithm

The Greedy Randomized Adaptive Search Procedure (GRASP) is a two phase procedure in which both phases are performed multiple times. The first phase consists of generating a random initial feasible solution based upon a random seed. The second phase uses some

form of a greedy local search to find the local minima. The solution is added to a solution candidate list. The process is repeated through a pre-defined maximum iteration value. Each iteration starts with a different random seed, leading to a presumably different initial solution.

Festa & Resende [43] provide a survey of the usage of GRASP in timetabling problems. Ribeirio & Urrutia[98] applies GRASP to the TTP problem. The approach is used by Hadjidj & H. Drias [50], and Burke et al. [19] for examination timetabling. Moura A. & Scaraficci [82], and Souza et al. [106] combine GRASP with tabu search in course scheduling.

The algorithm is an iterative approach that will run for a set number of times depending upon the problem instance. In each phase, an initial solution is generated, spawned from a random seed. This solution becomes the input for a local search effort to find the best solution, given the initial solution.

The pseudo-code for the algorithm, shown in figure 2.5 calls these two phases *Generate_Initial_Solution* and *Apply_Local_Search*. Both these phases are problem specific and hence could involve a variety of techniques. The solution S` represents the best solution to date, and f(S`) is the value of the solution that we seek to minimize. The variable *n* is the desired number of iterations and seed is a random number used by the *Generate_Initial_Solution* to create a solution.

```
Do (n times) {
        S  = Generate_Initial_Solution(seed)
        S`  = Apply_Local_Search(S)
        If f(S`) < f(S)  then   S = S`
}
```

Figure 2.5 – GRASP Algorithm Pseudo-code.

The solution *S* would be the best found solution for the algorithm.

2.3.4 Simulated Annealing

Simulated Annealing is another example of an approach that allows the acceptance of a solution that does not immediately improve the objective function. The approach developed by Kirkpatrick et al. [63] and generalized by Connelly [24] is driven by a temperature, which cools as iterations are made and changes in the objective function. When the temperature is hot, the approach is more likely to accept a move that does not improve the solution. As the temperature cools, it is less likely a non-improving move will be chosen. At this point, the approach becomes similar to the hill climbing approach.

The approach “cools” at a rate given by a function accepting the objective function difference between the current and proposed solution, and the current temperature. A

high temperature increases the probability of acceptance of moves decreasing versus this probability at low temperatures A difficulty with this approach is determining an appropriate cooling schedule, given the problem characteristics and available computing resources.

Simulated annealing is widely used in both timetabling and sports scheduling problems. Simulated Annealing was used in course timetabling by Abramson [1], Melicio et al. [80] and Abramson et al. [2]. Nandhini & Kanman [85] provide a survey of simulated annealing based approaches for the course timetabling problem area. Within examination timetabling Thompson & Dowsland [107] [108], and Mansour & Timiny [74] have applied simulated annealing. Anagnostopoulos [6] , Lim et al. [71] and Van Hentenryck & Vergados [115] all use simulated annealing to generate solutions for sports scheduling.

The pseudocode for this approach is presented in figure 2.6:

```
S  = Generate_Initial_Solution(seed)          //Initial Solution
Set Initial Temp T and Cooling Rate R         // input parameters
while (Temp  > 0 ) {                          //Current energy level high enough
                                              //to continue (may be max iterations)
        S` = Random(Neighbor(S) )
         If f(S`) < f(S)  then   S = S`       //accept solution if improvement
        Else
        If  exp(((f(S`) – f(S)) / T) < Random)
                then   S = S`                 //If  exp of difference between the
                                              //two solutions divided  by current T
                                              // is probable, accept solution
        T = T – R                             // Cool the current environment
}
```

Figure 2.6 – Simulated Annealing Algorithm Pseudo-code.

The solution *S* would be the best found solution for the algorithm.

2.3.5 Tabu Search

The tabu search was proposed and expanded by Glover [46] [47] as an alternative approach which enhances the local search of the greedy algorithm. The tabu search first enables a move to be selected when the move does not improve the objective function. This approach enables the search to move through more of the entire search space. However, allowing moves that do not improve the objective function can lead to cycling. Tabu search has been widely employed all areas of timetabling. Di Gaspero & Schaerf [36], White & Xi [117], Kendall & Hussin [58], White et al. [118] address examination scheduling while Costa [27], Hamiez & Hao [51] and Hertz [55] apply it to sports scheduling. Di Gaspero & Schaerf used the tabu approach for both the course scheduling[35] and TTP problem[34], discussed in detail in later sections of this paper.

The tabu search incorporates a list of recently used moves or solutions depending upon the application. The members of a tabu list can be used again for a period of time known as the tabu tenure. These tabu list members should be overlooked during the search for a set of amount of time or moves. An aspiration though can be used to consider moves or solutions on the tabu list. A common aspiration criteria is the solution is better than the previously best known solution.

The tabu search terminates upon reaching a maximum number of iterations, or a pre-defined number of iterations since the last improvement. The pseudocode for this approach uses a Done() function which indicates the search should end due to time considerations or a certain number of iterations. The *Tabu* function represents the tabu list and the *Asp* function the aspiration list. The tabu search is done in conjunction with

other local search method. This pseudocode assumes a function *Location Search* which determines whether a solution is accepted.

```
S  = Generate_Initial_Solution(seed)                    //Initial Solutio
while (Done() is false ) {                              //Current energy level high enough
                                                        //to continue (may be max iterations)
S` = Random(Neighbor(S) – Tabu(S) + Asp (S))            // select neighbor ignoring Tabu list
                                                        //  but considering Aspiration list
If LocalSearch(S`)  then   S = S`                       // select solution via search method
Add Solution to Tabu list                               //add solution to prevent cycling
   For all Solutions in Tabu list {                     // remove tabu list elements
     If tabu_tenure exceeded , remove Solution from tabu list
    }
}
```

Figure 2.7 – Tabu Search Algorithm Pseudo-code.

The solution *S* would be the best found solution for the algorithm.

2.4 INTEGER PROGRAMMING

Local Search algorithms can be built using customized software, integer programming, and constraint programming tools. Customized software provides the flexibility necessary to incorporate specialized heuristics in the search algorithm. However, this software is both challenging to build, and demands extensive development lead times to achieve even a simple case result. The options of integer programming and constraint programming can be implemented via a tool that models the problem. Integer programming (IP) generally implies modeling all events and constraints into variables with a value of zero and one. One variation of integer programming, “mixed integer programming”, allows the variables to have any integer value. Constraint programming allows for any value, including string values, to be assigned to the variables. The

constraint specification is also more versatile, providing the problem modeler with additional capabilities.

2.4.1 Integer Programming Related Research

Integer programming (IP) has been used in a wide variety of timetabling problems. The zero-one approach has been used by AfikBakir & Askop [4], and Daskalaki et al. [30] in university timetabling problems. These research problems look to find a feasible solution, given a set of constraints about courses, rooms and instructors. Burke et al. [18] looks to find the best solution to university timetabling problems which seek to minimize a set of penalties for violating soft constraints.

Sports scheduling has also been addressed by approaches based upon integer programming. One of the first to use integer programming was Fleurent & Ferland [44] in scheduling the North American National Hockey League. Ribeiro & Urrutia [100] and Duran [41] later used integer programming to schedule professional soccer leagues in South American countries. Some approaches combine integer programming with iterative search techniques as demonstrated by Guedes & Ribeiro [49], and Knust & Lucking [64]. These approaches contain a restart feature which enable integer programming models to be executed multiple times. The next section provides an example integer programming problem and solution.

2.4.2 Integer Programming Example

A standard and simplistic timetabling problem is scheduling examinations when a proctor is required for each examination. The proctor is required to have necessary qualifications

to proctor a particular examination. Table 2.1 contains a “1” in the cell when a proctor is qualified to give the exam.

Exam/ Proctor	Adams	Boyle	Carle	Dodge	Elgin
Math	1				1
Physics	1		1		1
Chemistry		1		1	
Biology	1	1	1		
English					1

Table 2.1 Sample Proctor / Exam Relationship.

The zero and one integer programming approach would use linear algebra techniques to form a diagonal of “1”s in the table. The result is shown in Table 2.2.

Exam/ Proctor	Adams	Boyle	Carle	Dodge	Elgin
Math	1				1
Biology	1	1	1		
Physics	1		1		1
Chemistry		1		1	
English					1

Table 2.2 Sample Proctor / Exam Solution.

The resulting matrix indicates that Adams should proctor Math, Boyle-Biology, Carle-Physics , Dodge-Chemistry and Elgin-English.

2.4.3 Integer Programming and Constraint Processing

Recently, integer programming has had great competition from a similar approach known as constraint processing. Both methods seek to formulate a problem in terms of an objective function and a set of constraints. The integer programming approach describes the constraints in terms of equations consisting of variables, each usually having a zero or

one value. Utilizing linear algebra functions, the sets of equations are solved to meet the objective function. Constraint programming also has variables, labeled as decision variables. These variables can take on both numeric and string values. Constraints can be specified with logical operators as well as equations. The constraint processing engine does not rely on linear algebra techniques, but rather a variable assignment process that utilizes concepts such as consistency checking and backtracking. Constraint processing is discussed further in section 2.5.

2.5 CONSTRAINT PROGRAMMING

Timetabling can be viewed as an exercise in constraint satisfaction. Constraint satisfaction is a formalized method to describe a problem and evaluate its solution. This method defines a set of decision variables, each with a possible set of values called the domain. The decision variables, taken together, represent a solution to the problem. In a timetabling problem, the slots holding the events are the decision variables.

Constraints are also defined. Constraints describe the permissible relationships between the events and the resources involved in the problem. A constraint may specify that an event must be assigned to a particular set of slots. An example constraint type would be a student only able to attend a class on Tuesday. A constraint may involve multiple events and resources. For example, not permitting a student to attend classes in consecutive slots is this type of constraint. The variables and constraints together represent the “model” of the problem.

Constraint satisfaction methods can be implemented by a variety of techniques. One of the first techniques used was integer programming. This technique involves describing all events and resources as variables, whose domain (allowable set of values) is integer. The constraints are then modeled as a set of mathematical equations involving these variables. Together the variables and constraint equations form a model. When the model is executed, integer values are assigned to the variables to satisfy the requirements of the equations. When a set of values assigned to the variables make all the equations true, a feasible solution has been found. The values of the variables represent the solution. These values often need to be translated from the model back to the original problem for solution publication.

Christiansen [23] defines the constraint satisfaction problem (CSP) by

(V,D,C): V is a finite set of variables $\{X_1,X_2, \cdots ,Xn\}$,

$D = D\times D\times \cdots \times D$ is the corresponding domains of n variables,

C is a finite set of constraints $\{C_1,C_2, \cdots ,C_r\}$,

where each constraint is defined by

$C_l(X_{l1} ,X_{l2} , \cdots ,X_{lt}) \subseteq D$,

that is, a set of all the legal t_l-tuples on variables $X_{l1} , \cdots ,Xlt$.

CSP differs from IP in that it allows *ad hoc* constraints, which may be given in any form (e.g., inequality constraints containing quadratic forms and all-different constraint stating that the values of variables in a given set V are all different). Because of this flexibility,

the formulations by CSP are usually much more compact than those of integer programming models.

CSP permits two types of constraints; hard and soft. A CSP model seeks a solution that minimizes the total penalty for violating soft constraints while satisfying all the hard constraints. The constraint definitions provide detailed relationships, among the decisions variables (and possibly other variables), that must be met. The objective function specifies the value of the solution. The objective function would incorporate the soft constraints within its evaluation.

2.5.1 Constraint Satisfaction Example

The recent expansion of constraint satisfaction usage has been spurred by the sports scheduling problem arena. Henz [52] [53] [54], Benoist et al. [12] , Schaerf [103] and Russel & Urban[101], have done groundbreaking work in this area. The common element in this arena is the scheduling of a round robin tournament. A single round robin tournament (SRRT) can be described as a league of set T with *n* teams (where *n* is even) to be scheduled, where each team plays each other team one time. A set of periods P (usually a week) exists such that an opponent must play a team in that week. Briskhorn [14] describes a generic version of this problem that also considers "breaks", which involve consecutive home or away games. A model of the SRRT problem is:

$$\text{Minimize} \quad \sum_{p \in P} \sum_{i \in T} \sum_{j \in T : j \neq i} c_{i,j,p} \; x_{i,j,p}$$

such that:

$$(1) \quad \sum_{p \in P} (x_{i,j,p} + x_{j,i,p}) = 1 \quad \forall_{i,j} \in T : i < j$$

$$(2) \quad \sum_{j \in T} (x_{i,j,p} + x_{j,i,p}) = 1 \quad \forall i \in T, p \in P$$

Where $c_{i,j,p}$ is the cost of team *i* playing team *j* in period *p*; and $x_{i,j,p} = 1$ if team *i* plays team *j* in period *p*, for a set of teams *T* and a set of periods *P*. Constraint (1) guarantees that each team will play each other one time. Constraint (2) ensures that each team plays every period.

There are no requirements in the problem about the number of home (or away) games for a team. Assuming, the cost of $X_{i,j,p} =. X_{j,I,p}$, we need only look at matchups where $i < j$. Table 2.3 presents sample cost data for a tournament of six teams:

Xi,j	Period 1	Period 2	Period 3	Period 4	Period 5	Average
(1,2)	10	20	30	40	50	30
(1,3)	15	20	25	30	35	25
(1,4)	40	20	60	80	100	60
(1,5)	50	40	30	20	10	30
(1,6)	10	10	10	10	10	10
(2,3)	5	10	15	20	25	15
(2,4)	60	50	40	30	20	40
(2,5)	5	10	15	20	25	15
(2,6)	20	15	20	15	20	18
(3,4)	10	10	10	10	10	10
(3,5)	30	25	20	10	5	18
(3,6)	20	20	20	20	20	20
(4,5)	10	5	10	10	10	9
(4,6)	30	40	50	40	5	33
(5,6)	5	10	15	20	25	15

Table 2.3 Cost Data for Sports Scheduling Example

The mathematical formula to minimize cost represents the objective function. This cost may be travelling distance, or an accumulation of soft constraints violations. This function will increase as assignments are made to $x_{i,j,p}$. The $c_{i,j,p}$ are costs that are fixed and input to the model. There are two constraints presented in the model. The first

constraint forces each team to play each other team exactly one time. However, this model does not look for an even distribution of home and away games seen in many real world sports scheduling problems. The second constraint ensures that each team will play each week.

2.5.2 Constraint Programming Tools

Constraint programming tools, such as ILOG-CPLEX, is an engine to execute a constraint satisfaction model. Its job is to assign decision variables in such a way that minimizes the objective function. The constraint programming tool uses many steps to develop its best solution. The model is created and run to cause assignments to decision variables must be made. As assignments are made, care must be taken to ensure all hard constraints are satisfied. The next sections look more closely at the assignment process and the related steps in executing a constraint programming model.

2.5.2.1 Variable Selection and Assignments

The first step in variable assignment is to select which decision variable should be assigned a value from its domain. An instance of the $X_{i,j,p}$ variable would be chosen and assigned the value of one. In the constraint programming form of x[i,j] = p, a particular x[i,j] would be chosen for assignment. In the latter case, after the variable is chosen, a value of its domain would be selected for assignment. Initially, all values of P would be in each x[i,j] variable. A selection rule would decide which of the domains values to use for assignment. This selection rule may be selecting the domain that minimizes the objective function.

A simple strategy for selecting the next variable is to choose the variable with the smallest domain. In the SRRT problem above, the x[i,j] with the smallest domain represents the game that has the fewest periods eligible for assignment. As assignments for a particular team are made, remaining games for the team have fewer period choices, since teams play only once per period. Several other strategies can be employed in selecting the next variable assignment. Dechter [33] refers to "look-ahead" strategies that are used to select the next variable or its assignment value. The decision whether to accept a variable and its assignment value may be based upon its impact on the other decision variables. A "forward-checking" strategy ensures that this variable's assignment will not cause any other decision variable to have a domain size of zero. A domain size of zero would imply that that decision variable has no acceptable value and hence no solution will be generated. Other types of forward checking involve "consistency " discussed in the next section. Davenport & Tsang [31] discuss repairing consistency during checking.

The assignment of a variable triggers the "constraint propagation" process. The assignment, in concert with the constraint, will reduce the domain of other decision variables. This may result in some decision variables having a domain size of one. Variables of domain size one indicate there is only one eligible assignment that can now be made for this decision variable. In this instance, the values will be immediately assigned. This propagation assignment will trigger yet more domain reductions and further assignments. Within our example problem, the x[i,j] will have a domain size of

one when there is only one period in the tournament where both team *i* and team *j* are still available to play.

2.5.2.2 Consistency

A set of decision variables are consistent if all the values within their domains could be used in a solution. If the assignment of a domain value will cause another decision variable to have a domain of zero, then the set is viewed as inconsistent. Any variable assignment may cause the domain of other variables to be reduced and raise the possibility of inconsistency. Hence, after each variable assignment, there is an opportunity for consistency checking at various levels.

The first level of checking is termed “arc consistency”. This consistency check ensures that the assignment of any value remaining in the decision variable’s domains will not cause an inconsistent or non-feasible solution. Arc consistency checking can be done by temporarily assigning a remaining domain value of a decision variable, performing constraint propagation, and verifying that all decision variables have at least two elements in their domain. This process is done for each value in every domain variable. This checking is also referred to as simply propagation.

Additional consistency checking is termed “path consistency”. This consistency checking ensures that all the assignment values of the decision variables will result in an arc consistent network. Hence, it involves two levels of checking for each domain variable. Path consistency requires temporarily assigning a remaining domain value of a decision variable, performing constraint propagation to temporarily modify the decision variables,

and then performing arc consistency again on the temporary set of decision variables. Path consistency consequently applies arc consistency (along with propagation) twice on a temporary set of decision variables. Some constraint programming approaches use n-consistency, where arc consistency is performed n times on a temporary set of the decision variables.

2.5.2.3 Backtracking

Consistency checking may reveal that a proposed assignment value for a decision variable cannot lead to a feasible solution. If this value is the only remaining domain value for the decision variables, then previous assignments were inconsistent (or the problem has no solution). Backtracking involves revisiting previous decision variables and selecting a different value the second time. The execution of backtracking requires the decision variables to be revisited in the exact reverse order of the original assignments. When a decision variable is reached through backtracking, its assignment value is removed from its domain, and another value is selected.

Backtracking can result in “thrashing”, where a very early decision variable assignment in the sequence of assignments is inconsistent. This early assignment is not re-evaluated until all the values of the all the decision variables in subsequent assignments have been evaluated. If limited search time is available for backtracking, then the thrashing may prevent finding solution. One approach to reduce thrashing is “backmarking”. This enables the constraint programming program to backtrack several assignments to a designed location in the assignment sequence.

2.6 SUMMARY

Constraint programming models are used extensively to implement various strategies of search, such as depth-first search and simulated annealing. By controlling the variable assignment order, the model developer can force a certain strategy in exploring the search space.

The drawback to constraint processing is the requirement to declare variables and related constraints for all aspects of the problem. For example in sports scheduling, home and away patterns are developed to help manage the assignment of the game location. This variable in conjunction with the decision variable enables the model to alternate home and away games for a team, when possible. However, this approach introduces another n (number of teams) by g (number of games) matrix to the model. This proliferation of variables hinders the model in generating solutions for instances with a high number of teams.

The tiling approach described in the next chapter is able to generate solutions by combining assignments into tiles based upon conjectures. These conjectures may come from the personal experience of the scheduler or from a problem specific heuristic.

CHAPTER 3 - TILING APPROACH

3.1 INTRODUCTION

The tiling approach is hybrid of constraint processing and local search techniques. The approach consists of two phases: tile placement (phase I) and local search (phase II). The first phase places tiles, which are groups of logically related decision events, in the timetable. This process uses many of the constraint processing steps described in Chapter 2. This tiling activity creates an initial solution, which may or may not be feasible. The second phase of local search, swaps events within the timetable to achieve a feasible solution and improve its quality.

This chapter defines the basic steps of the tiling approach, and contains an algorithm for the tiling technique for timetabling problems. The next four chapters present applications of this approach to specific problems.

The fundamental idea of the tiling approach is that relationships between events can be established and utilized prior to the placement of any event within the timetable. The relationships are documented by the creation of a set of events in a tile. These tiles may consist of a set of events appearing consecutively in the timetable, or events that should not share resources. The particular relationship used to create the tile varies by problem. For example, a tile may be a set of courses that need to be scheduled in the same period, or a set of sports contests to be played in consecutive weeks.

In a research environment, we frequently create relationships between events by using various algorithmic techniques, which are devoid of practical business knowledge. In the real world, timetabling practitioners understand probable event relationships through experience. Tiling enables the practitioner to input this valuable expertise into our solution approach, without constraining the eventual solution.

3.2 PHASE I OF TILING

The first phase of the tiling is a constructive approach to create an initial solution. Depending upon the demands of the problem, this solution is not necessarily feasible. Initial solutions that are not feasible are made feasible during the second phase (local search).

The goal of the first phase is to provide a high quality initial solution in order to lessen the reliance on the second phase. The initial solution achieves this through a fast and efficient placement of multiple events referred to as tiles. The events in the tiles have inter-relationships due to a high probability of their proximity in assignment within the ultimate solution. The next sections detail the steps involved in creating the initial solution.

3.2.1 Tile Creation

The tile is a set of events to be assigned within the timetable simultaneously. Each event within the timetable is referred to as a block. A block, in constraint processing terms, would be a decision variable. Each block must be placed in the timetable.

Each timetabling problem requires its own distinct definition of the relationship represented by the tile. To illustrate the tiling approach, the solution to a single round-robin tournament is presented in table 3.1 for four teams. There are at least two methods for constructing tiles for this problem. The first would be to create the best schedule for a given team *i*. All *Xi,j,k* variables for a particular i would be analyzed. The set of *p* and *j* values that would minimize $\sum_{p \in P} \sum_{j \in T} c_{i,j,p}\ x_{i,j,p}$ would be chosen for the tile. The second option would be to select the best games for each period. The objective function $\sum_{i \in T} \sum_{j \in T: j \neq i} c_{i,j,p}\ x_{i,j,p}$ would be minimized to find the best set of games for each period. In this discussion, we choose to select the latter option and select the best matchups for each visiting team. This approach will also yield tiles of different lengths, since each team plays a distinct number of away games. The tiles created for this problem are shown in Table 3.1.

Team	**Matchups and Periods**	**Cost of Tile**	**Average Block Cost**
1	(1,2,1),(1,3,3),(1,4,2),(1,5,5),(1,6,4)	75	15
2	(2,3,1),(2,4,5),(2,5,2),(2,6,4)	50	12.5
3	(3,4,2),(3,5,5),(3,6,2)	35	11.67
4	(4,5,2),(4,6,5)	10	20
5	(5,6,1)	5	5

Table 3.1 Sample Tiles.

3.2.2 Tile Costing

A tile is assigned a "cost" after its creation. The cost reflects the increase to the objective function of the failure to place the tile. When a tile cannot be placed it is "broken" into its component blocks. The assignment of the three blocks, independently of each other, will increase the objective function (assuming a well constructed tile).

The cost of a tile is the difference between its contribution to the objective function and the increase to the objective function of assigning each block of the tile independently. Since neither measurement may be fully known before the assignment process, it is not possible to state with certainly the true cost of its breakage. However, a value that is a good predictor of the magnitude of the increase to the cost function of breaking the tile can be assigned to the tile. In the example problem presented in this chapter, we would use the average cost of xi,j over all periods, noted in the last column. For example, the cost of assigning the tile for Team 3 is thirty-five as shown in Table 3.1. This is the tile's contribution to the objective function. We compare this against the total of each block's average objective function contribution. These averages are ten for $X_{3,4}$, 18 for $X_{3,5}$ and twenty for $X_{3,6}$, giving a total of forty-eight. The tile assignment cost of thirty-five is subtracted from the total of tile breakage (48) to give thirteen as the "cost" of the tile. By comparison, consider the tile for Team 4. Its assignment cost is ten, the assignment of each block using the block average is forty-two, giving a total tile cost of thirty-two. This value for Team 4 is over twice the value of Team 3, suggesting that Team 4's tile should have priority in placement over Team 3's tile.

3.2.3 Tile Placement

The tiles are placed into the timetable after the creation and costing steps. The tiles are placed in the timetable by giving each of the blocks within the tile an assignment. In a course scheduling problem the assignment might be a period of the class for a student. Within sports scheduling, it may be selecting the week for a match. In our sample problem, tile placement is simply updating the timetable to reflect the information (opponents and period) embedded in each block of the tile.

A key consideration in tile placement is the ordering of the tiles. Tiles with the highest cost are given priority and placed before the lower cost tiles. In the sample problem, we would assign Team 4 schedule in the timetable before assigning Team 3, due to the higher cost of Team 4's tile.

Some timetabling problems may enable the tile to be reconfigured during processing to consider additional slots. The reconfiguration would involve reordering of the blocks within the tile. If the tile does not include the actual period, as seen in the sample problem, the tile may be able to placed at various slots within the timetable. For example, in a course scheduling problem, a tile may represent three courses, A,B,C to be scheduled consecutively during the day. This problem has a timetable of slots, where each slot represents a period in the day. The tile placement process would select which period would be the starting period for the three assignments. If period 1 was selected than the courses would be scheduled in periods one through 3. If period 4 was selected, the assignments would cover periods 4 through 6.

The tile may also have its blocks rotated to avoid violating constraints. In the course scheduling example, the tile of A, B,C courses may be assigned with the courses in a different order , e.g. C,B,A.

3.2.4 Consistency Checking

Consistency checking is performed after the placement of a tile. Each block in the remaining tiles and broken tiles has a domain of eligible slots for placement without constraint violation. The assignment of any tile to a set of slots in the timetable affects the domain of all the remaining blocks. For example in our sample problem, the scheduling X_{12} in period 3 would remover period 3 from X_{14}, since Team 1 can only play one game in a period. The reduction of X_{14}'s domain could result in the domain having size zero, one or a value greater than one.

When a block reaches a domain size of one, the block can be assigned to the timetable. This process is referred to as constraint propagation. This process is iterative as each block's assignment reduces other blocks' domain leading to possibly more propagatioin If a domain size of a block to be scheduled is zero, then there are no slots for this block within the timetable. At this point, the last tile placement cannot be made. The tile will be broken into its blocks and used in the final step of Phase I , discussed in the next section.

3.2.5 Backtracking and Phase I completion

The final step of Phase I begins when all tiles have been placed in the timetable or have been broken into blocks. These blocks are placed into the timetable with one or more constraints relaxed. The relaxation allows the blocks to be assigned, regardless of the existence of a feasible solution. Any hard constraint violations introduced in this step is addressed in the Phase II local search. Examples of constraint relaxation include allowing course to be in impermissible slots or permitting invalid sequences of games in a sports scheduling problem.

The placement of these blocks is done using the concepts of constraint programming discussed in Chapter 2. The ordering of the blocks is done by sorting the blocks based upon domain size. Blocks with a low domain size are assigned first. After each assignment, all remaining block domains are modified, and the blocks re-ordered. When a block's domain is of size one, the block is immediately assigned. A block with a domain size of zero causes a backtrack of the last block assignment. If the previous block cannot be assigned to any slot in the timetable, backtracking is performed again. If a feasible solution (given the constraint relaxation), cannot be found, then the last tile placement is backtracked and this Phase I completion step is exercised again.

3.3 PHASE II – LOCAL SEARCH

Phase II involves a local search where assignments are swapped to remove any remaining constraint violations and reduce the objective function. The phase involved defining a set of actions that will modify the solution. These actions are dependent upon the particular

timetabling problem. Generally, the action involves the reassignment of a set of variables to different slots in the timetable.

There are many methods outlined in Chapter 2 to perform local search. Our work is centered on the tiling which focuses on developing high quality initial solutions. Hence, we only use the steepest descent hill-climbing search during Phase II in our problem approaches discussed in Chapters 4 through Chapter 7. This method provides a very fast and efficient way to take advantage of the tiling and provide a high quality solution.

3.4 TILING ALGORITHM

The following tables and figure provide a detailed description of the tiling algorithm. Table 3.1 lists the variables used within the algorithm and table 3.2 defines set of functions used by the algorithm. Table 3.3 contains the logic steps of the tiling algorithm.

Variables	**Definition**
b	individual block / event assignment
t	individual tile consisting of multiple event assignments
t[n]	tile t placed at slot n of the timetable
{B}	set of Blocks
{CS}	Set of candidate solutions
{T}	set of Tiles
BackTrackStack	stack of blocks assigned in the solution.
S	A solution , may or may not be feasible

Table 3.2. Tiling Algorithm Variables

Function	Use
assignBlock(S,b,n)	assigns block b at slot n within Solution S.
assignTile(S,t,n)	assigns tile t at slot n within solution S.
best({S})	returns a solution with the lowest objective function value.
chkBlock(S,b,n)	boolean value indicating that block b can be places at slot without violation of non-relaxed constraints.
breakTile(t)	returns the blocks of tile t
chkTile(S,t,n)	boolean value indicating that tile t can be placed at slot n without constraint violation.
f(S)	returns the value of the objective function for a solution.
highestCost(T)	returns the tile with the highest cost of breakage.
lowestDomainSize(S)	returns the block with lowest domain size.
rotateTile(t)	return a tile with different slot assignments each block within the tile.
solution(S,m)	returns a solution after applying move m on solution S.

Table 3.3 Tiling Algorithm Functions

ALGORITHM PROCEDURES

```
Create Tiles (problem specific) to create tiles and initialize {T}
{B} = null ; S = null
While {T} not empty
        t = highestCost({T})
        {T} = {T} – t
        For each slot n
                If checktile(S,t,n) then assignTile(S,t,n)
                Else
                        t′ = rotateTile(t)
                        checktile(S,t′,n) then assignTile(S,t′,n)
                Else
                        {B} += breakTile(t)
End while
For each b in {B}
        b.lastslot =0
While {B} not null
        b = lowestDomainSize({B})
        flag = false
        For each slot n starting b.lastslot + 1
                If chkBlock(S,b) then
                        assignBlock(S,b,n)
                        b.lastslot = n
                        push b onto BlockTrackStack
                        {B} = {B} - b
                        i = true
                        exit loop
        next
        if not flag then pop BackTrackStack to {B}
End while
For each type of Move
        {CS} = null
        For each move m
                S′ = solution(S,m)
                {CS} += S′
        Next
        S′ = best({CS})
        If f(S′) > f(S) then
                S = S′
        Else
                Stop
next
```

Figure 3.1 - Complete Algorithm for Tiling

The following three chapters demonstrate how the tiling approach can be applied to problems defined by the research community. Each of these chapters contains a description of the problem, an explanation of the application of tiling, and a comparison with results from the research community. Chapter 7 describes a new problem to the research community. A formal definition and sample instance are provided for this problem. The chapter also includes an explanation of the application of tiling to a real world instance of the new problem.

CHAPTER 4 - CONSTRUCTING INITIAL NEIGHBORHOODS TO IDENTIFY CRITICAL CONSTRAINTS

Recent course scheduling competitions have seen solution approaches which construct an initial solution quickly, and then employ a local search to improve the solution. With the use of different seeds, this process is repeated, searching for the best solution. Solutions with constraint violations provide little guidance on which constraints to relax in order to produce a better quality solution. The tiling approach seeks to construct several high quality initial solutions and analyze their characteristics which enables us to predict the relative success of the local search phase. With this capability, sets of initial solutions can be generated with selected constraint relaxations, leading to a prediction of which constraint relaxation can most improve the final solution, leading to a good quality solution.

4.1 INTRODUCTION

The Metaheuristics Network sponsored an International Timetabling Competition in 2003 [90], involving a course scheduling problem. This competition was followed by a second competition, with different tiers focusing on variations of course scheduling. The objective is to assign courses to a day / time / room, avoiding “hard” conflicts and

minimizing “soft” constraints. An evaluation function is used to determine the value of the solution based upon the soft constraint violations.

The majority of approaches to these problems involve a two-phase approach. The first phase establishes an initial solution, which is usually free of hard constraints (though not an absolute requirement). The second, and more intensive phase, is a local search, where course assignments are swapped or moved to other time periods, reducing the soft constraint violations and improving the quality of the solution. Burke and Newall describe such an approach in [17]. Kostuch [65] reports the best results with a two phase approach.

The creation of the initial solution is typically highly stochastic and hence may or may not serve the local search phase well. Both phases are repeatedly performed with different seeds, to obtain the best overall solution. The initial solution represents little more than a starting point for the second phase.

Our approach has two main objectives. The first is to develop a method to predict whether an initial solution will serve as a good base for the local search phase. The second objective is to use this prediction capability to generate sets of initial solutions, where each set relaxes a different constraint of the problem. These sets of initial solutions, differing only by their constraint relaxation, can identify the constraints most affecting the solution quality. The scheduler, could then re-consider a course’s constraints, or accept the existing soft constraint violations. McCollum [77] speaks of a

similar goal stating the need for "*identification and comparison of key dataset characteristics and potential linkages with the likely best search approach to be taken*".

4.2 PROBLEM DEFINITION AND INITIAL SOLUTION GENERATION

This problem was created for the course timetabling problem of the Second International Timetabling Competition [93], held in 2007. This problem, within Tier 3 of the competition, consists of instances with courses (each having multiple sections), curricula associated with sets of courses, course availability within weekly time-periods, and room capacities. Each instance has the following constraints:

- Two sections cannot be scheduled in the same room during the same time period;
- Teachers cannot teach two sections during the same time period;
- Courses with the same curriculum can be scheduled during the same time period;
- All sections of a course cannot be scheduled during an unavailable time period;
- Sections should not violate room capacities;
- Sections of a course with the same curriculum should be in a time period next to another section in the curriculum;
- The course sections must run over a minimum number of days
- All course sections should share the same room.

The first four constraints are hard constraints, while the last four are soft constraints. Violations of the soft constraints increase the evaluation function.

Our approach considers each course as a tile, with each section being a block in the tile. A similar approach was used by Kingston in [60][61]. In Kingston's problem a "form" is the equivalent of a course, while a "meeting" is equivalent to a section. The sections are positioned in a greedy and constructive manner within the scheduling grid, according to that course's constraints. Bar-Noy and Moody showed tiling to be an effective approach to establishing a partial solution in [9]. Each section is placed in a "slot", which is a combination of a room/time period. Each tile placement minimizes the soft constraint

violations of the course, given the previously scheduled tiles. During the tiling step, the room capacity constraint is treated as a hard constraint. As more courses are scheduled, remaining courses become increasingly more difficult to place, without violating hard constraints. The courses are ordered for assignment based upon their scheduling "difficulty", a function of the number of sections, curricula and unavailable time period constraints. Different orderings are generated through the introduction of a random factor, which can lower a course assignment in the order, delaying the tiling of the course. This stochastic factor enables various initial solutions to be constructed, though the tiling process is constructive.

At the tiling step conclusion, remaining unscheduled sections, due to the room capacity constraint, are scheduled. If unscheduled sections still exist, this course assignment order is not included in the set of initial solutions.

4.3 INITIAL SOLUTION ANALYSIS AND LOCAL SEARCH PHASE

Each initial solution is analyzed for its support of the local search phase. Several local search approaches are available, and we utilize a simple swapping method. The swapping method involves switching the slots of two sections, or moving a section to an unused slot. Our implementation calculates the impact on the evaluation function when swapping each section with every other section. The swap causing the largest reduction is actually performed, breaking ties randomly. The schedule is then reevaluated for positive swap candidates. This process repeats until no swaps can be made to improve the evaluation function. This hill-climbing method does not make a swap, unless there is an immediate benefit to the evaluation function.

Our objective is to analyze the quality of the initial solutions and predict whether they will lead to a high quality final solution. This approach embeds a common constraint processing technique of "look-ahead strategies" described by Dechter [33]. In this strategy, the remaining domains of unassigned variables are used for ordering purposes. We extend this concept to evaluate the overall value of the solution in terms of reaching a high quality final solution.

Our analysis of an initial solution looks at several characteristics of the solution:

- **Solution Value**: the evaluation function value of the initial solution before the local search phase
- **Total Swap Count**: The total number of possible swaps in the solution which have a positive effect on the evaluation function.
- **Swapability Value**: This section prorates the reduction in the evaluation function over all possible moves. For example, if a section can be moved to two time periods, each swap's value is multiplied by 0.5 and added to this value.
- **Movement Value**: This section counts the total number of slots a section can be assigned to, given the section currently in that slot is removed from the time period. A section's movement to another slot is often restricted by other sections within the slot's time-period having the same curricula or having the same instructor. If section A in time-period 1 is swapped with section B in time-period 2, we need to be sure section A does not violate a hard constraint with any other course (other than A) in time-period 2. We need not worry about hard constraint relationships between A and B, since they will be in different time-periods after

the swap. This value indicates how tightly constrained the schedule is with respect to hard constraints.

We seek to create a prediction factor based upon a function of the initial solution characteristics discussed above. The following tables in figures 1 and 2 show the initial solution analysis for three instances from the competition. Each instance has had 20 initial solutions generated, differing only by the assignment order of the courses in the tiling step. For each initial neighborhood, the four characteristics above are presented, along with the final solution value after the local search phase.

SEED	INITIAL SOLUTION VALUE	MOVE-MENT VALUE	SWAP-ABILITY FACTOR	TOTAL SWAP COUNT	FINAL SOLUTION VALUE
1	193	14596	1451	1542	16
2	225	14240	2173	1434	64
3	286	14591	1415	1868	74
4	276	14474	1665	1584	78
5	230	14224	1523	1590	22
6	315	14439	2119	1494	88
7	224	14089	2414	2014	25
8	230	14132	1582	1892	56
9	263	14371	2036	1558	28
10	331	14283	2557	1886	28
11	286	14258	2668	2228	52
12	253	14251	2148	1976	59
13	225	14259	1437	1842	24
14	270	14295	1951	1912	28

15	271	14305	2551	1650	45
16	221	14438	1861	1736	47
17	260	14316	3586	2144	32
18	227	14049	1390	1862	54
19	186	14576	1476	1288	23
20	304	14287	2320	1558	25

Figure 4.1 Results From Instance 1

SEED	INITIAL SOLUTION VALUE	MOVE-MENT VALUE	SWAP-ABILITY FACTOR	TOTAL SWAP COUNT	FINAL SOLUTION VALUE
1	514	112138	2289	23996	179
2	567	112705	3342	25782	173
3	596	113062	3443	26312	182
4	573	111655	2676	24900	189
5	528	112097	2709	23354	188
6	563	112405	1900	25818	185
7	562	112398	2422	25256	181
8	616	112111	3824	26426	197
9	572	112091	3589	25098	185
10	562	113192	2979	25778	195
11	601	112817	2623	27270	200
12	558	112666	2951	26218	184
13	564	112066	3366	25936	204
14	615	112523	3037	26264	199
15	566	112059	3512	25566	216
16	556	113078	2723	26060	205
17	520	111665	2365	24954	181
18	574	112342	3038	25164	186
19	564	112296	2449	25182	207
20	562	113190	2979	25578	165

Figure 4.2 Results From Instance 8

4.4 RESULTS

Our results, presented in Figures 4.1 and 4.2, show a high correlation between the Initial Solution Value and the Final value. In the three instances shown, as well as for the other instances not reported here, we note that the best final solution value came from one of the top five initial solutions.

Other factors can be used to separate the set of high quality initial solutions. Consider seed 1 and 19 in instance 1, shown in Figure 4.1. Seed 19 had a better initial value, but less number of total swaps available. Comparing seed 19 to seed 20, the final solution results were nearly identical although there is a 70% difference in the initial solution value. In this instance, the swapability factor was 50% higher in seed 20, indicating potential in performing swaps to improve the evaluation function. The best final solution of instance 8 came from the use of seed 20. However that seed's initial value of 562 was 48 higher than seed number 1. This greater degree of improvement in the local search phase could possibly can be attributed to the higher movement and swap factors within seed 20 versus seed 1.

Our results demonstrate, for certain seeds, that we have an ability to predict, within a range of a probability distribution, the relative quality of the final solution. This work can viewed as a prediction factor, that can evaluate a set of initial solution. This prediction factor, can then be used to select the best solution to move into Phase II.

.

4.5 CONCLUSION

Scheduling professionals in the field need high quality schedules and guidance on schedule constraint relaxation to achieve a useable schedule. This chapter shows how a

tiling method to quickly construct quality initial solutions, and assigning to each initial solution a factor predicting the quality of the final solution.

Chapter 5 examines the tiling method associated with another competition by the Metaheuristics Network.

CHAPTER 5 - Construction of Initial Solutions for a Course Scheduling Problem Using Tiling

A previous competition of the Metaheuristics Network involving course scheduling produced many solution approaches that constructed an initial solution, and then improved that solution using a local search. As the competition (which would also be the case in the real world) was time constrained, a fast and efficient initial solution was crucial to allow more time for the local search phase. In this chapter, we propose a tiling technique that can quickly construct a high quality initial solution. The tiles represent conjectures on the best placement of courses before any assignment process. The tiles are used to generate a solution that can be used as a good starting point for the following local search procedure. We show how our tiling technique creates the initial solution quicker and with comparable quality, to the initial solutions of the best known approaches.

5.1 INTRODUCTION

The Metaheuristics Network sponsored an International Timetabling Competition in 2003 [90], involving a course scheduling problem. Several competitors provided solution approaches, all of which relied upon the generation of an initial solution, followed by a local search phase. The search phase used techniques such as simulated annealing [65], tabu search [35], and the great deluge algorithm [20]. The creation of the initial solution was done randomly or employed a straightforward prioritizing mechanism. Our work focuses on a construction algorithm, utilizing a tiling technique that provides a feasible initial solution that can serve as a foundation for a local search. The tiles represent sets of classes that should be scheduled together to minimize the objective function. We create

the tiles in a method specific to the competition problem. However, in the real world, the tiles would represent the deep business knowledge and experience of past scheduling exercises. Professional scheduling personnel with years experience develop conjectures about the final schedule, based upon past successes. We look to incorporate this knowledge within the tiles.

Our approach creates the initial solution, using only a small amount of computational resource. This allows the majority of the available time to be dedicated to the local search. We show that ourapproach is able to create the initial solution quickly, is scalable for large instances, and does provide solution quality competitive with the best known approaches for this problem. In addition, our tiling approach provides interim results for the user, enabling a scheduler in an industrial environment to modify available scheduling resources before the entire schedule is produced.

5.2 PROBLEM DEFINITION

The competition presents a university course timetabling problem. It consists of a set of events to be scheduled into forty-five timeslots, across five days and nine periods. Each event must take place in one of the rooms provided for each instance. Rooms are constrained by room size and features, eliminating the possibility of certain events from being held in a given room. Each event is attended by a set of students. Other hard constraints of the problem include students only attending one event in any one time period, and at most one event scheduled in the room for each day and period. Meeting all hard constraints for each event constitutes a “feasible” solution.

The problem also contains soft constraints. Violation of soft constraints increases the evaluation function, which we seek to minimize. Hence, solutions with the least soft constraint violations represent the best solutions. The soft constraints are:

- A student has a class in the last slot of the day.
- A student has more than two consecutive classes.
- A student has a single class on a day.

A review of the entries submitted to the competition solutions show that most approaches were in two phases. Typically, the first phase builds an initial feasible solution. The second phase performs some form of search looking to move previously scheduled events. Burke & Newall describe such an approach [17].

In the top four entries, the method for creating the initial solution varied as well as the search algorithms. Kostuch [65] had the best set of results. Since Kostuch presented solution scores and times, for each of the two phases, we focus on these results in this thesis. This enables us to compare our results to his initial solution creation. The Kostuch approach uses simulated annealing (referred to as phases III and IV in his paper) to achieve the best solution quality. His approach in establishing the initial solution was to place events into timeslots, and then attempt to assign rooms. If a feasible solution could not be obtained, then the algorithm randomly unscheduled a set of events and tried again. Fifty runs with different seeds were used to achieve the best feasible solution.

Other approaches, such as Burke & Bykov [15] and Bykov[20], used a modified Brelaz (saturation degree) algorithm. Events were analyzed for the number of timeslots available. Events with the lowest number of timeslots were scheduled first. DiGaspero &

Shaerf [35] used a random placement of events for the initial feasible solution, while Courdreau [28] did not create a feasible solution before the search phase. Their approaches rely solely on the search phase to not only improve the solution but also create a feasible solution. Muhlenthaler & Wanka [83] relies on a set of partial timetables that permit their insertion strategy to develop a feasible solution. They identify events which can placed, if all conflicting events can be successfully moved to other timeslots. This approach relies on Kempe chain insertion described by Mauritsius [75]. Finally, Fen et al. [56] uses a fitness function within constraint programming to find a solution to a similar problem.

5.3 OUR APPROACH

Our approach is a constructive and deterministic algorithm that creates an initial solution to support the more intensive search phase. Kingston [62] discusses how a phased approach is effective in time-table construction. The algorithm is based upon the concept of tiles, which are a series of variable assignments. In this problem, a tile would represent all events in all rooms for two consecutive time periods. The goal is to create tiles that have minimal constraint violations and still enable a feasible solution to be reached. We create two tiles per day, covering four periods in each day. Excluding the last period, which violates a soft constraint in the objective function, the tiles cover half the entire schedule for an instance of four hundred events. The creation of the tiles must be balanced by the requirement to create a feasible schedule. The events not included in a tile must be scheduled within the remaining open “slots” (a room and timeslot combination) of the schedule. We aim to create good quality tiles while preserving enough flexibility within the remaining events to construct a feasible schedule.

The constructive approach has relevance in producing the initial solution, which a local search algorithm can operate upon. We seek a good quality initial solution that can be found within several seconds (a small percentage of the allowable competition time), and provides a good starting point for the search algorithm. Cormen discusses building upon initial solutions in [26].

We use tiling as a means to create the initial solution in a deterministic manner. Bar-Noy & Moody [9] demonstrated that tiling was an effective approach for the Traveling Tournament Problem [109] to quickly develop a solution within ten or less percentage points of the best known solutions. The tiles for that problem were a set of games that were to be scheduled sequentially within a team's overall schedule. Tiles were placed in a team schedule, as long as hard constraints were not violated when considering other tiles in the other teams' schedules. Kingston [60] [61] also used tiles for a set of classes in a high school scheduling environment.

For our problem, a tile represents a set of events that will be scheduled in the same day and in period x and period x+1, where x is a period from 1 to 7. The events in the tile relating to the same period must not share any students, since this would violate the hard constraint of a student attending two events in one timeslot. If the tile can also avoid violating the consecutive classes soft constraint, the tile provides a "break" in the student's schedule. Since no student attends more than one event in a tile, the student has

a break between period *x* and period *x*+1. After the tiles are created and placed, the remaining events are placed with a simple greedy algorithm to form the feasible schedule.

The tiling approach can also be viewed as creating a "macro event" defined by McCollum [77]. The macro event is a collection of events that can be associated together. This association may be because a student would take them as a block, or in our case, that the events are independent of each other. The ability to recognize relationships between events (beyond the obvious constraints) is key for tile or macro event usage.

5.4 TILING CONCEPT

A "tile" is a set of up to 2*n* (*n* is the number of rooms) events. In the Metaheuristics Network Competition, the value of *n* was usually ten. Considering the events of a tile in two sets of size *n*, the first set was a set of events that could co-exist in the same day and time period. Hence, the first *n* events that had an event-event relationship of zero were chosen for the tile. This first set of events is to be placed at period *x* on a given day. After the first set of events for the tile was chosen, a second set for period *x+1* was selected. This second set also needed the event-event relationship to be zero in order not to violate the hard constraint of a student taking two classes in a period. However, the second set also seeks to minimize the number of students taking classes in both the first set (period x, where x is any period from 1 to 7) and the second set (period x+1). This is calculated by summing the event-event relationship values between all events in the first and second set. The lower this sum, the more likely students are to have a "break" in their daily schedule. In our approach, we only allow a maximum of nine students to be taking an event in both periods of a tile. This means the

most students will have a break in classes that will help reduce the evaluation function by minimizing violation of the consecutive classes constraint. Figure 5.1 illustrates this point.

This information can be used to determine a key relationship used in the problem; an event to event relationship. This relationship is helpful in two ways. We give a value to the relationship equal to the number of students required to attend both events. For hard constraints, the relationship specifies which events can share timeslots that have the same day and period. All event-event relationships values between all events in the same day and period timeslot must be zero. For soft constraints, the relationship provides the level of potential violation of the consecutive classes constraint for a student. If two events have an event-event value that is positive and the events are scheduled in consecutive timeslots, these events may lead to a soft constraint violation, given the events scheduled around the pair, for the student.

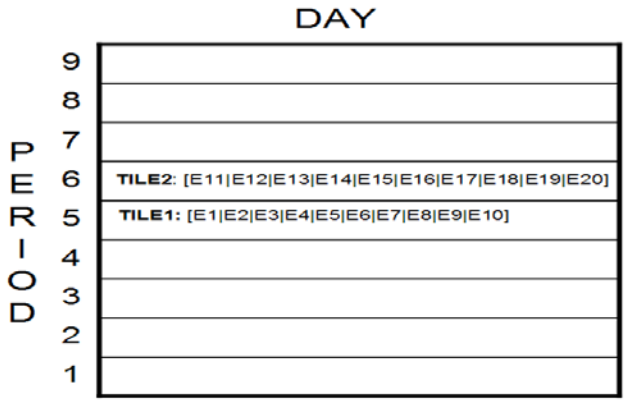


Figure 5.1- Tiles slotted in periods 5 and 6.

Once all tiles are created, they are placed in specific timeslots within the schedule. Unlike the Traveling Tournament Problem addressed by Bar-Noy & Moody[9], the placement of the tiles is independent of each other. In the Traveling Tournament Problem, tiles consisted of a set of games. Placement of a tile could conflict with previously scheduled games. Bar-Noy and Moody used a backtracking mechanism to move the tile in this situation. Within our problem, the placement of tiles in unique timeslots cannot cause a hard constraint violation, since no hard constraint involves events from two different periods.

For each instance our approach generated *r* tiles sets (discussed in section VI), each consisting of ten tiles where *r* is the number of rooms The tiles of the best tile set were then placed in the schedule as shown in figure 5.2.

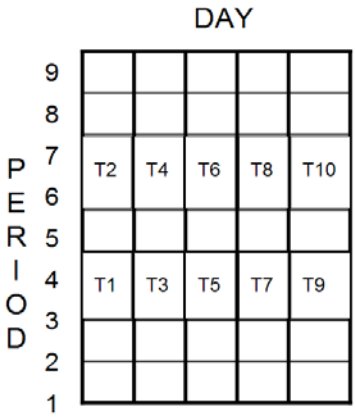


Figure 5.2 - Tile placement within the schedule.

We selected the above tile placement to help minimize the number of students violating the "taking more than two classes consecutively" constraint. A tile placed at period *x* represents a break in all student schedules, for that day, between period *x* and *x*+1. For a student to cause a soft constraint violation for consecutive classes, the student must take

classes in periods 1-2-3 or 2-3-4 or … up to periods 7-8-9. Placing the tile at period 3 ensures that no student takes all classes in periods 2-3-4 and 3-4-5. Placing another tile at period 6 also ensures that no student takes all classes in periods 4-5-6, and 5-6-7. A student can cause two constraint violations by having four classes in a row. Placement of the tiles at period 3 and period 6 ensures no student is in this category. Hence, the placement of the tiles at periods 3 and 6 minimized the combination of periods in which a student could have 3 or 4 consecutive classes. We have not experimentally explored the concept of moving tiles to different periods. However, placing the tiles closer together would provide a longer time period block where no breaks could occur, while leaving a time period block (possibly up to five periods), where more than three consecutive classes could occur. In the latter scenario, students could have more than three classes consecutively, while in our approach, we have tried to limit the maximum number of consecutive classes for a student to three.

In order to complete the schedule after tile placement, we need to place the remaining events not contained in a tile. A major factor affecting the ability to schedule remaining events after tile placement, for an instance, is the flexibility of events to be in multiple rooms. Instances where many events can only be scheduled in one room provided more of a challenge in scheduling the remaining events. For example, consider the situation where the ten tiles have been created, covering 20 periods, or half of the schedule. If there are twenty-one events remaining that are only allowed in room *r*, then the events cannot be scheduled without using the last period since only 20 open slots are left for room *r*. An approach to address this dilemma is to use more events only allowed in room *k* (where *k* is a valid room number) in the tiles, leaving events that can be scheduled in

room *k* AND other rooms, in the set of remaining events. Forcing this situation may denigrate the quality of the tile, measured by students taking a class in both periods of the tile. For example, event A may not share students with all other events in the tile, but is not used since it can be assigned to multiple rooms, leaving more flexibility for the later scheduling step. Using a more constrained event may be necessary in order to achieve a feasible schedule, though it may share a few students with other events in the tile.

Our approach focuses on this trade-off of tile quality versus feasible solution generation. We generate r sets of tiles by first including only events with one room allowable, then another set of tiles with events of one or two allowable rooms, up to events allowing all (or *r*) rooms. Key indicators discussed later enable us to pick the set of tiles that maximizes the quality of the tile while allowing for the remaining events (events not in a tile) to form a feasible solution.

5.5 PRE-PROCESSING STEPS

The Metaheuristics Network competition problem is presented by a series of files that indicate the following relationships:

- Rooms to Features.
- Features to Events.
- Students to Events.

By transitivity, rooms to events can easily be calculated. This information can be analyzed to determine the key relationship in the problem – event conflicts between students (event to event conflicts). The event-event relationship is helpful in two ways. We give a value to the relationship equal to the number of students required to attend both events. For hard constraints, the relationship specifies (by being greater than zero)

which events can share timeslots that have the same day and period. For soft constraints, the relationship provides input on the potential violation of the three or more consecutive classes for a student. If two events have an event-event relationship greater than zero and the events are scheduled in consecutive timeslots, then these events may lead to a soft constraint violation.

The final step of the preprocessing phase is to calculate the degrees of each event. The degree of an event is defined as the number of room/period combinations that are possible for the event. Initially, this value is $t*n$ where t is the number of time periods and n is the number of rooms, since each event can be placed in any timeslot, if no other events are scheduled. In the next phase, as events become scheduled, the degree for events will decrease, as possible slots are used and become unavailable. For example, if event A can be scheduled in room 3, 4, 5 of a period, initially day 1, timeperiod 1 will add 3 degrees to event A. If event B is scheduled in room 3 of that period, and there are no students taking events A and B, then the degree of A reduces by one. If event C is scheduled in the period in room 7, and there is a student taking events A and C, then we must decrease the degree of event A by 2. This is because since no slots are now available in that timeperiod for event A, because of the hard constraints.

5.6 TILE SETS AND SCHEDULE COMPLETION

Following the pre-processing step above, "tilesets" are created for the instance. A "tileset" is a set of ten tiles, which each event in the tileset is allowed to be scheduled in k rooms. Tileset_k, refers to a tileset. Tileset_3 would have ten tiles where each event is allowed to be placed in at most three rooms. Tilesets are created for k = 1 to r. After each

tileset is created, a straightforward greedy algorithm (described below) is used to place remaining tiles in a schedule. Only tilesets that can produce a feasible schedule are chosen for analysis.

For all tilesets producing a feasible schedule, the "placement value" is calculated for the tileset. This value, as an equation is: (the number of events contained in all the tiles) - (tiles scheduled in the last period) *r. This value is an excellent indicator of the tileset that will produce the best initial solution, after the tuning step, which is discussed in the next section.

The number of events in a tileset, referenced in the first part of the formula, may not always be 20*r. For example, the tileset including only events allowed to be scheduled in one room, may not have enough events to fill ten tiles. Moreover, the events may conflict with each other to prevent ten perfect tiles of 2r events to be generated. For example, the more events contained in the tiles, the less likely the consecutive class soft constraint will be violated. Periods containing incomplete tiles, will be completed during the greedy scheduling step. The greedy scheduling step does not consider the soft constraints, hence leading to more possible violations of them.

After the tiles are created and placed, the remaining events (not contained in a tile) need to be scheduled. This scheduling task is carried out by a greedy approach, where the next events are chosen based on the lowest remaining slots. Events with the fewest slots remaining, considering only hard constraints, are placed in the first available day and

period, starting with day 1 and period 1. Depending upon the events chosen for the tiles, the remaining events may not be able to form a feasible solution for a tileset.

Our approach creates up to *r* tilesets for each instance. The first tileset has only events with one allowable room. This tileset places highly constrained events (based on allowable rooms) into the tileset, so these events will be scheduled within a tile, thus before the greedy approach step above. If the remaining events are highly constrained, the greedy approach may fail to reach a feasible solution due to room availability. By selecting only these highly constrained events for the tileset, the greedy approach will more likely find a feasible solution. However, more soft constraints may be violated within the tileset since fewer events are available for selection. When our approach considers events with more than a few allowable rooms for a tileset, the tileset has fewer soft constraint violations. However, the remaining events (not in any tile) may be too constrained to reach a feasible solution. In all instances in the competition, at least one of the tilesets led to a feasible solution. For all tilesets of an instance that led to a feasible solution, we selected the tileset with highest placement value for further tuning of the initial solution.

5.7 TUNING THE SCHEDULE

After the best tile set is chosen and greedy scheduling performed to achieve a feasible solution, tuning is carried out. The first tuning attempts to move any event in the last period (which violates a soft constraint) to the first eight periods of the schedule. This is done through an exhaustive pass through the schedule. The event currently in last period is analyzed to find all periods that do not have a student conflict for the event (ignoring

the event with the same room as the last period event). For each period allowing the event, the last period event is swapped with the event currently in the same room. The swapped event is the “candidate event”. This candidate event is then compared to all open slots in the schedule for placement. If the candidate event can be placed in an open slot, then the last period event can be moved to the candidate event’s period, and the candidate event can be moved to the open slot.

Figure 5.3 illustrates this process. In the figure, event 104 is the last period event. Event 134 is the candidate event. A search is done on all periods, where Event 104 replaces the event currently in room 3. If no hard constraints are broken, then the event replaced becomes the candidate event, in this case event 134. An attempt is made to schedule this candidate event in any remaining open slot in the schedule. In Figure 5.3, period 4 and room 4 hold an open slot allowable for event 134. The example implies that event 134 must be allowable in rooms 3 and 4. The example illustrates the impact of having events allowed in multiple rooms.

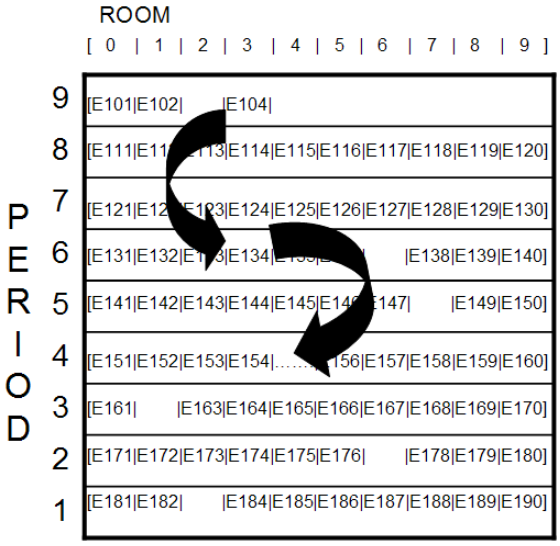


Figure 5.3- Moving Last Period Events into the Schedule.

Next, a second type of swapping step is performed. All the events in one period replace all events in another period. If the swap reduces the evaluation function value, then the swap is retained. In order to constrain the resources in this step, our approach limits the number of comparisons. We look at each period, starting with day 1 and period 1 and find the best period for a swap. We perform the swap and then move to day 1 period 2. This approach guarantees at most forty swaps will be performed, since there are forty time periods.

5.8 THE COMPLETE ALGORITHM

This section shows the complete algorithm described in the section above. The next two tables define variables and functions, followed by the main and subroutine procedures.

The variables used in the algorithm are show in figure 5.4. The functions are presented in figure 5.5. The pseudo-code for the various routines are found in Figures 5.6 through 5.9.

Variables	Definition
Allow_rooms	The number of different rooms allowed for the event.
Best_placement_ratio	The best placement_ratio found among the tilesets for an instance.
Best_tileset	The tileset with the best placement ratio.
Event_degrees	The number of available slots in the schedule for an event.
Last_period_events	The set of event currently in period 9.
Max_allow_rooms	The maximum value permitted for an event's allow_rooms for a tileset.
Number_rooms	The number of rooms involved in the instance
Placement_Ratio	A metric of a tileset predicting the final solution score of the schedule.
Room_Event(Room,Event)	Boolean value indicating if Room is allowable for the Event.
Schedule_Event(	Specifies the number of students

Event1,Event2)	attending both Event1 and Event 2.
Students_consec	Number of students taking classes in both periods of a tile.
Tile_events	Set of events contained within a tile of the tileset
TileSet	A set of ten tiles, each tile containing up to twenty scheduled events in consecutive periods.
Unscheduled_events	Set of events which have not been assigned a day and period

Figure 5.4 - Algorithm Variables.

Functions	**Use**
Available_Room (Event, Day,Period)	Returns true if able to find an empty room in the slot at Day/Period, that is allowed for this Event.
Calc_Score	Calculates solution score of schedule by counting soft constraint violations
Swap Periods(A,B,C,D)	Move all events in Day A, Period B to Day C, period D, and vice versa.

Figure 5.5 - Algorithm Functions.

```
   MAIN ROUTINE
call PROC-INITIALIZE-ARRAYS
set Best_placement_ratio to 0
for Maxallowrooms goes from 1 to Number_rooms
  store all events in  Unscheduled_Events
  call MAKE-TILESET
  call SCHEDULE-TILESET
  call GREEDY-SCHEDULE
  for all events in Unscheduled_events
     call EXCHANGE-EVENT-U
  end for
  for all events in Last_period_events
     call EXCHANGE-EVENT-L
  end for
  set Placement_ratio to  Tile_events +  (Last_period_events*10)
  if Placement_ratio > Best_placement_ratio
    set   Best_tileset to Tileset
    set Best_placement_ratio to placement_ratio
end for
call SCHEDULE-TILE-SET with Best_tileset
best_score = Calc_Score
 for i goes from 1 to number of days in week
   for j goes from  1 to number of penalty-free periods in day
     for m goes from 1 to number of days in week
       for n goes from 1 to  number penalty-free periods in day
           call SwapPeriods(i,j,m,n)
           set score = Calc_Score
           if best_score < score then
                Schedule events in day/period i,j into m,n
                Schedule events in day/period m,n into i,j
         end for
       end for
     end for
  end for

SCEHDULE- TILE-SET
    For all events  in Tile_set, label it A
      place A in schedule
    For  all events in Unscheduled_Events, label it B
      for  I goes from 1 to number of days in week
       for  J goes from  1 to number penalty-free periods in day
          for events  scheduled in Day I, Period J, label it C
            if  Student_Event(B,C) is false
              if Available_Room(A,I,J) then
                add 1 to Event_Degrees
```

Figure 5.6 - Algorithm for the Main Routine

```
GREEDY-SCHEDULE
   For   events  in Unscheduled_Events, label it A
    for I = 1  to number of days in week
     for  J= 1 to number of penalty-free periods in day
        for events  scheduled in Day I, Period J, label B
          if  Student_Event(A,B) is false
            if Available_Room(A,I,J) then
              place event A in Day I, Period J
      end for
     end for
    end for
  end for
```

Figure 5.7 - Algorithm for the Greedy Schedule Routine.

```
EXCHANGE-EVENT_U
   For  events A  in Unscheduled_Events, label it A
     for I goes from 1 to  number days in week
      for  J goes from 1 to number penalty-free periods in day
        for K goes from 1 to A event's allow_rooms
          labe event in Schedule(I,J,K) as B
              for M goes from 1 to to number days in week
                for  N goes from 1 to number penalty-free
       periods in day
                    for events scheduled in Day M, Period N,
                        label it C
                          if  Student_Event(B,C) is false
                            if Schedule(M,N,event B's room)
      is empty
                                 place event A in Day I, Period J
                                 place event B in Day M Period N
                  end for
                end for
              end for
          end for
        end for
    end for

EXCHANGE-EVENT_L
   For   events in Last_Period_Events, label it A
      Call EXCHANGE_EVENT_U using L
   end for
```

Figure 5.8 - Algorithm for the Exchange Event Routines

```
PROC-INITIALIZE-ARRAYS
   read InstanceDefinition_File
   for each event , label it A
     for each event , label it B
       if A and B share a student
            Student_Event(A,B) is false
          else Student_Event(A,B) is true
     end for
   end for
   for each event
     for I goes from 1 to Number_rooms
        if Room_Event(I) is true
           add 1 to Allow_Rooms
     end for
   end for
   for each event
      Set EventDegrees to 40 (number of day/periods)*Allow_Rooms
   end for
```

Figure 5.9 - Algorithm for the Process Initialization of Arrays Routine.

```
Make -TILESET
  set TileSet to null
  for events in Unscheduled_event sorted by event_degree
  label it A
    if Allow_rooms < Max_allow_rooms
      for Events in Tileset, label it B
       If Student_Event(A,B) is false OR
           Students_consec < Number_rooms
            if Available_Room(A) then
               place event in TileSet
      end for
  end for
```

Figure 5.10- Algorithm for the Make TileSet Routine.

5.9 RESULTS

We present our results in Figure 5.11. We show our initial solution values for all twenty instances compared with Kostuch approach's initial solutions. The first column identifies

the problem instance, the next two columns present the percentage of allowable computation time for the competition used by our approach and our initial solution score. The fourth and fifth columns show these same values as published by Kostuch. The Kostuch result is used since results were published after creating the initial solution, and before the more extensive local search phase. Also, the Kostuch scores are the best known scores. Times for our approach and the Kostuch approach are measured in terms of percentage of allowable competition time. This allowable time is based upon the hardware environment, and hence provides a way to measure different environments.

The sixth column shows the portion of the time used by Kostuch, that we used to obtain a better or similar result. For six instances we used 1/3, or less of the resources as Kostuch to generate the initial solution, yet our initial solutions are within 5-15% of the Kostuch score. In five other instances, we achieved a better score, yet used only between 50% to 90% of the resources. In only one instance (8), did we use more time (2%). However our initial solution was 7% better for that instance.

Inst	%Our Time	%Kost Time	Our Score	Kost Score	Our% of Kost time	Score Diff %
1	2.78	5.90	266	**245**	0.47	8.57
2	2.96	4.00	**217**	219	0.74	-0.91
3	1.39	10.90	273	**262**	0.13	4.20
4	3.89	10.10	560	**350**	0.39	60.00
5	3.43	12.20	449	**392**	0.28	14.54
6	3.98	4.10	377	**347**	0.97	8.65
7	3.43	4.60	439	**365**	0.74	20.27
8	3.98	3.90	**252**	271	1.02	-7.01
9	2.87	3.40	266	**249**	0.84	6.83
10	3.06	10.40	272	**261**	0.29	4.21
11	2.13	8.00	301	**259**	0.27	16.22
12	2.50	10.90	255	**246**	0.23	3.66
13	3.89	9.80	393	**301**	0.40	30.56
14	4.35	6.90	394	**368**	0.63	7.07
15	3.24	5.20	341	**325**	0.62	4.92
16	2.41	3.00	**243**	274	0.80	-11.31
17	3.24	15.90	501	**387**	0.20	29.46
18	2.31	4.50	**196**	225	0.51	-12.89
19	5.19	7.40	410	**367**	0.70	11.72
20	3.15	3.50	**284**	330	0.90	-13.94
AVG	3.21	7.23	334	**302**	0.56	10.69

*Figure 5.11 – Results, best results **bolded**.*

Analyzing the table, we see our approach is twice as fast on average per instance. We used only 56% of the comparative resources used by the Kostuch approach. Yet using half the resources, we averaged a score that was within 10% of this approach. Discounting the three highest events, our approach comes within 4% of the Kostuch initial solution scores.

The explanation for some of the variances can be readily seen in Figure 5.12. This table shows some of the key interim indicators, mentioned in our algorithm, that highlight the characteristics of the instance.

Instance	Tile Events	Unscheduled Events	Last Events	Place-ment value	Score Diff %
1	193	13	10	93	8.57
2	194	13	7	124	-0.91
3	174	9	8	94	4.20
4	167	25	19	-23	60.00
5	168	10	8	88	14.54
6	148	14	8	68	8.65
7	191	22	6	131	20.27
8	138	4	2	118	-7.01
9	197	17	8	117	6.83
10	129	4	3	99	4.21
11	186	22	17	16	16.22
12	147	12	11	37	3.66
13	179	20	16	19	30.56
14	119	2	1	109	7.07
15	140	1	1	130	4.92
16	199	13	4	159	-11.31
17	167	18	16	7	29.46
18	193	6	6	133	-12.89
19	140	6	5	90	11.72
20	173	4	0	173	-13.94
AVG	167		7.80	89	10.69

Figure 5.12 - Key Indicators per instance

This table identifies the instance in the column 1. For each instance, we see the number of events that were contained in the ten tiles within column 2. Columns 3 and 4 present the number of events that were handled during the tuning processes. The Unscheduled Events are those events still unscheduled in the first eight periods after the greedy scheduling step. Column 4 represents the events scheduled in period 9 violating the soft constraint. The placement value is the number of tile events (column 2) – the value of number of last period events (column 4) * *r*. This placement value provides an insight into how difficult it is to create a feasible schedule while minimizing the use of the last

period. In instances where our score differences were the highest (instances 4, 13, and 17), the placement value was very low. Our tiling approach is driven more to minimize the consecutive event soft constraint violation, versus the last period constraint violation. Hence, instances that demand a great deal of swapping to avoid the last period give us poorer results. Figure 5.12 clearly shows the last events soft constraint is the cause of our higher scores.

The placement value gives a sense of the success of the tiling approach for a particular instance. A low placement value, or even negative number, implies that the tiles are not very dense (many openings in tiled periods), and hence our approach is relying more upon the simplistic greedy scheduling step. When the placement indicator is high, the problem instance is able to take advantage of tiling benefits. For example, in instance 4, the placement value is negative and the solution score generated was poor compared to the Kostuch stochastic approach. While in instance 20, the placement indicator was extremely high and our approach produced a initial solution 14% better than Kostuch's initial solution. The key characteristic leading to these results is the distribution of allowable rooms across events. When room usage is constrained (many events trying to use the same room), the tiles have a lower quality in terms of density (events per tile period), and the consecutive classes soft constraint between the events in period *x* of a tile and *x+1* of a tile.

5.10 CONCLUSIONS

In this chapter we demonstrate that a tiling approach can generate an initial solution in half the time and of nearly equal value to the best known approach. The time savings are critical to allow sufficient resources for the phase II search. While the stochastic

approaches to initial solution generation ranges widely in both time and quality, the tiling approach is a predictable and a reliably fast method to generate high quality initial solutions. This deterministic approach is not dependent on multiple runs using different random number seeds and can scale easily. Our algorithm is polynomial, and hence, could handle larger instances. We do not claim, however, that in every instance, we could find a feasible solution in polynomial time. However, we were able to do so with the competition instances.

In our approach, a “tile” would represent the optimization of a small set of events. One possible view of a tile, for a real world problem, would be to consider the tile as a department schedule. Tiles would be placed into a master grid in order to create the initial neighborhood. A second phase of local search would resolve any conflicts. Hence, the tiling approach mimics the process used by many institutions in the real world.

Utilizing the tiling approach, we are able to incorporate all the business knowledge of the departments into the solution. These departments may have developed local schedules, for years and often know the best placement of courses for their own student population. The departments may also use historical schedules as a basis for creating the new schedule. Often, minimal perturbation is desired so that students and staff can plan, knowing that the new schedule is likely to closely mimic the previous one. The constructive tiling approach provides a more consistent solution over multi-year schedules, since the initial neighborhoods would be similar.

Also within industry, these problems will typically contain many more events and constraints. For example, in a large university, there may be several thousand courses, together with additional constraints such as faculty availability, travel time between courses, and curriculum attributes. Typically in a university, some levels of scheduling decisions are made at the department level, and these are translated to a master scheduling grid, where conflicts are resolved.

In academic problems, where there is no institutional background knowledge, the tiling approach loses some of its advantages. A tiling method must be developed that (near) optimally creates a set of events based upon some heuristic. However, in a real world problem, the heuristic can include the institution's previous experience/knowledge during the tile creation process.

The next chapter switches to a new problem area – sports scheduling. Chapter 6 discusses how the tiling approach is applied to the well research Travelling Tournament Problem.

.

CHAPTER 6 - TTP SPORTS SCHEDULING PROBELM

The Traveling Tournament Problem (TTP) describes a typical sports scheduling challenge. The TTP, which is based on the U.S. Major League Baseball (MLB) league, has specific instances with results available on the web. Several approaches have been proposed since the problem's creation. The "best" of these solution methodologies use extensive computational resources in local search activities to find high quality solutions. This chapter shows how using a tiling method that can produce good quality solutions, using a fraction of the resources documented in other approaches. The tiles represent conjectures about the relationship among particular games in the schedule, and form an initial solution for a local search phase. This thesis shows how these tiles are found within the best known TTP solutions to date. In addition, the tiles enable us to develop the actual MLB 2010 schedule, with its broader set of scheduling challenges.

6.1 INTRODUCTION

The Traveling Tournament Problem (TTP) is a double round robin tournament to be played by n teams over $(2n-2)$ periods or weeks, where each team plays in every period (we do not consider the "mirrored" version of the problem). The three constraints of the TTP are:

1) Maximum "**Road trip**" of three games: each team can play at most three consecutive games away from the team's home site before playing again at the home site. For teams beginning the season with an away game, it is assumed travel for that game would begin at the home site. Likewise, teams ending the season with an away game, would require to travel from that opponent's location to the team's home site to complete the season.

2) Maximum "**Home stand**" of three games: each team can play at most three consecutive games at its home site.

3) Repeater Rule: a team cannot play an opponent away in time period *k* and then home in time period *k+1*, or vice versa.

The TTP seeks to minimize the total distance traveled by all teams. A distance matrix defines the distance between each team, which is used to calculate distance travelled. For each road trip, with the total being the accumulated distance for all teams. The selection of opponents and their order within the road trip is critical, while home stands have no bearing on the distance calculation. Efficient road trips across all teams are more likely to yield good quality solutions to the TTP.

The TTP has served as a benchmark problem for sports scheduling over the past decade. Easton et al. [42] defines the problem, and the latest results are available at [109]. However, the TTP definition notes its simplification of the actual Major League Baseball. This simplification eliminates some key complexities of the MLB, notably:

Existence of unbalanced schedules – Teams in the MLB play teams within their division more frequently than teams in other divisions.

Unequal home and away games – Due to Inter-league play, teams do not play an equal number of home and away games against all teams. The actual MLB problem also introduces constraints of teams from the same city playing on the same day. Kendall [57] notes this constraint in scheduling the English football league.

These two simplifications enable many solution approaches to take advantage of round-robin tournament specific scheduling techniques for the TTP. For example, some approaches depend on a mirrored approach, which could not be applied to an unbalanced schedule. Other TTP approaches use simple tournament generators that also could not accommodate an unbalanced schedule. We propose an approach that can provide quality initial solutions whose components can be found within the best solutions. In addition, our solutions compete favorably with the best TTP tournament based solutions sets and are extendable to handle the real MLB problem.

6.2 RELATED WORK

The TTP has spawned a variety of methodologies to find good quality solutions. Kendall et al. [59] surveys this broad array of approaches. The work to date can be viewed in two phases. The first phase consists of traditional approaches including simulated annealing by Anagnostopoulos [6], Lims' [71] work on integer programming, the constraint programming approach by Leong [67] , and tabu search proposed by Di Gaspero & Schaerf [34]. These approaches use a single process to direct an intensive local search. Some search techniques include special heuristics as used by Shen & Zhang [105]. The second phase focuses on the use of parallel processing and server farms. Van Hentenryck & Vergados [115], and Araujo [7] use a parallel implementation of existing algorithms to improve on published results.

The majority of the above methods have focused almost exclusively on local search operations to produce the best solutions. The initial solution for the local search is usually

generated at random, and is not necessarily feasible. Hence the starting "solution" may be extremely far from the optimal solution, in terms of a distance measured by the number of "swaps", which are defined as moves between solutions.

Our approach seeks to take the results of related graph algorithms to form conjectures on the relationships among the games in the instance. We seek to predict which games will ultimately be in the road trips for a team. We model the road trips of the TTP as "tiles". The tiles with the highest "cost" of breakage (discussed in section 3.3) are used in the initial solution formulation. The tiles with the lowest cost are broken and scheduled game by game to fill out the initial solution. We then use a local search phase to remove any remaining constraint violations, and tune the initial solution.

This approach seeks to create an initial solution that is a feasible solution and is closer to the best solution than random initial solutions. The tiling method, proposed by Kingston [61] for course scheduling, creates an initial solution that has minimal hard constraint violations, and contains a core set of class assignments for a high quality solution. The concept of modeling the problem using away trips was previously considered by Trick [112]. A new variable in Trick's work is introduced to represent a road trip. The road trip variable is assigned individual games which immediately affects the value of the objective function. The road trip variable is continually modified to reduce the objective function. Hence many road trips are considered for a given team during the exercising of the model. This approach differs from our approach, in that we begin the scheduling

process with existing road trip (tile) formations. During our scheduling phase, we do not re-formulate a tile based upon other assignments. We may break a tile into its component parts, but these parts are not reassembled into another tile. Our approach is based on a fixed set of tiles.

6.3 PROPOSED APPROACH

Our approach is a two phase approach involving tiling, followed by a local search phase. The tiling phase creates an initial solution, which may not be feasible. The second phase (a local search) removes the hard constraint violations and improves the quality of the solution.

6.3.1 Tile Creation

Each tile contains "blocks" that represent individual games. A road trip of three opponents is considered as one tile, with three blocks. A team's schedule can be thought of as a series of tiles, with home games as spacers between the tiles. Figure 6.1 shows the scheduling grid and tiles for Team 1 and Team 2.

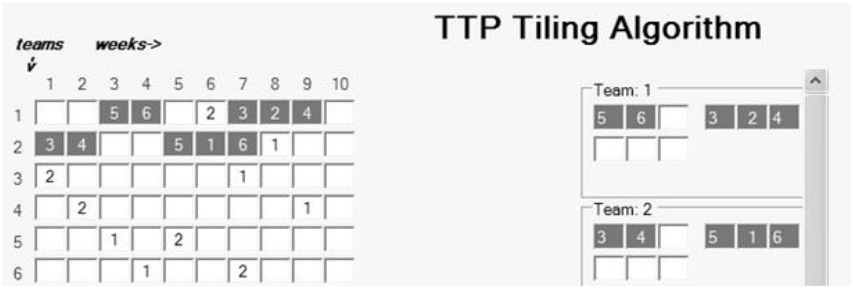


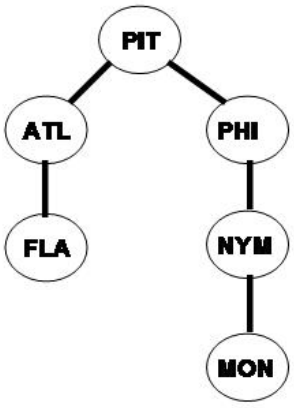
Figure. 6.1 Tile Placement for Teams 1 and 2 (shaded cells indicate away games).

For each team a set of tiles is created that seeks to minimize the distance traveled for a particular team, without taking into account any constraints involving other teams. These tiles are placed in a scheduling grid of n rows representing teams and $(2n-2)$ columns representing weeks.

As tiles are placed, other cells of the grid are filled in to maintain schedule consistency with the tile placement. When no more tiles can be placed, the remaining tiles are broken into their component blocks, and placed subject to TTP constraints. If all blocks cannot be placed, the consecutive home and away constraint is relaxed, allowing all blocks to be placed, albeit in an infeasible solution.

The creation of the tiles is carried out on a team by team basis. For each team a minimum spanning tree (MST) is created by using the Prim [26] algorithm. The algorithm begins with the selection of a root node, in our approach this is a team. All distance edges in the distance matrix, defined in the problem, are searched for the smallest edge. Each edge has a vertex of a team in the tree, and a vertex of a team not in the tree. This edge is then added between the two teams. For the first branch, we are finding the nearest opponent of the root team. The second edge is the nearest team to the root team, or its first opponent. The third edge is nearest to the root team, or the first two opponents. In general, the nth edge is nearest team to the base team or $n-1$ opponents within the tree. Edges that are added to an opponent will suggest that the two vertices or teams will be on the same road trip or tile. Two edges having the same root node as a vertex, suggest that the two

opponents of the root team will be on different road trips. Figure 6.2 presents an example MST for the team Pittsburgh (PIT) in the NL6 problem documented in [109].



DISTANCE MATRIX

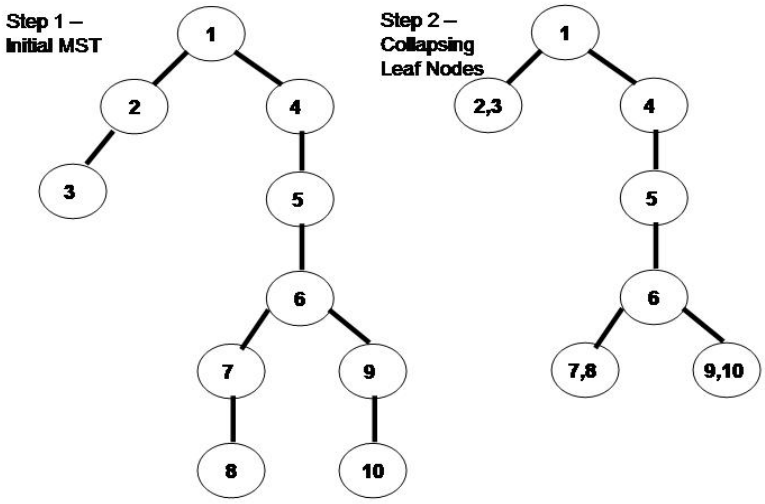
Home Team \ Away Team	ATL	NYM	PHI	MON	FLA	PIT
ATL	0	745	665	929	605	521
NYM	745	0	80	337	1090	315
PHI	665	80	0	380	1020	257
MON	929	337	380	0	1380	408
FLA	605	1090	1020	1380	0	1010
PIT	521	315	257	408	1010	0

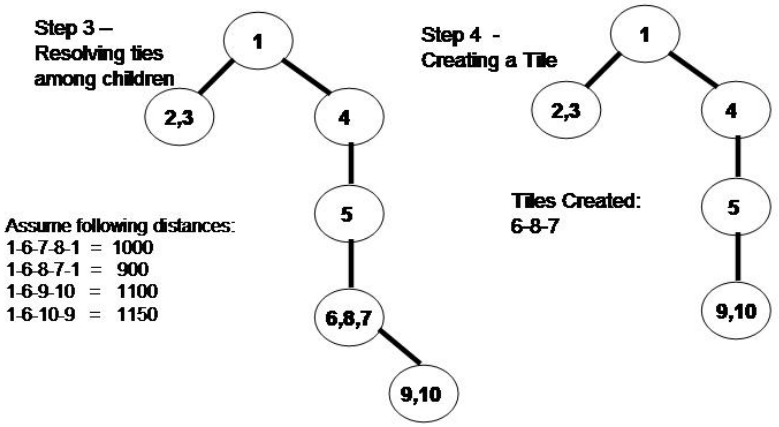
Example: Distance from Team NYM to Team PHI is 80 miles

Figure. 6.2 Minimum Spanning Tree (MST) for PIT in NL6 and the accompanying distance matrix.

Figure 6.2 suggests that the team PIT should have two road trips or tiles – one with ATL and FLA, and the other with PHI, followed by NYM and MON. If the team completes these two road trips, it will have travelled the optimal minimum distance using MST. This does not imply the league overall will have optimal minimal distance, but rather just this team.

We use a tree collapsing algorithm to create tiles from the tree structure. Separate trees are created with each tree having the root node of a team. The collapsing approach merges child nodes into their parent node. When the parent has three teams, a tile is created. When the parent must decide among its children, a greedy method is used, to pick the best set of three teams from the parent and children. Figure 6.3 provides a sample collapsing process.


Step 1 –
Initial MST
1
2
4
3
5
6
7
9
8
10
Step 2 –
Collapsing
Leaf Nodes
1
2,3
4
5
6
7,8
9,10


Step 3 –
Resolving ties
among children
1
2,3
4
5
6,8,7
9,10
Assume following distances:
1-6-7-8-1 = 1000
1-6-8-7-1 = 900
1-6-9-10 = 1100
1-6-10-9 = 1150
Step 4 -
Creating a Tile
1
2,3
4
Tiles Created:
6-8-7
5
9,10

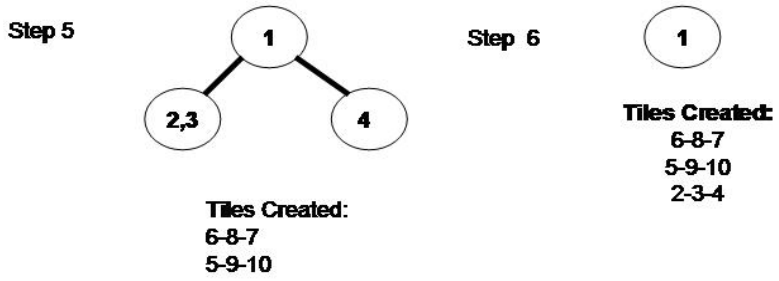


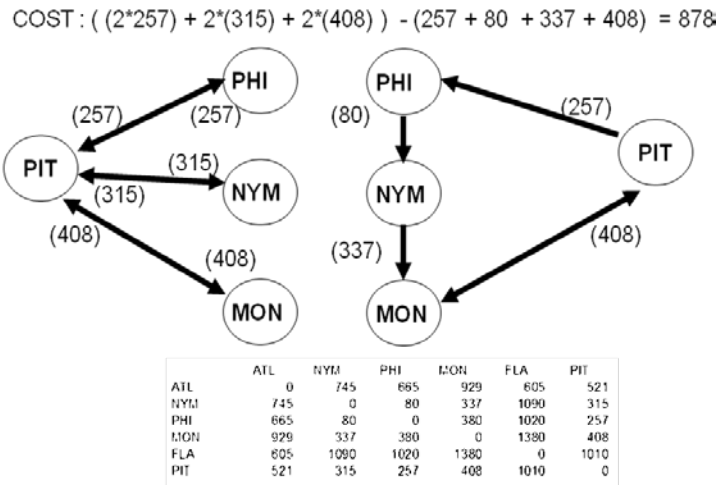
Figure. 6.3 Creation of Tiles through collapsing of the tree.

The three team tile creation process creates $\lceil((n-1)/3)*n\rceil$ tiles. Note that all tiles have three blocks, with the possible exception of the last tile created for the team. At the root node, if both children have a weight of two nodes, two tiles of two blocks are created for each child.

6.3.2 Tile Breaking Cost

Our approach generates a set of tiles, representing our conjecture of the best road trips for a team, before any placement in the schedule. Each tile is assigned a cost that indicates the effect of the breaking the tile. Breaking of a tile implies the team will need to travel home during the road trip and return to one of the cities in the tile to complete the road trip. The cost is assigned by calculating the difference between the team using the tile as a road trip versus visiting each team in the tile separately. Our approach suggests that tiles with higher costs should be included in the solution intact. While tiles with lower costs should be chosen to be broken into their individual games, when individual games are needed to obtain a feasible schedule. Tiles have a high cost when the distance to travel home and return to the road trip is greater. The games within a low cost tile are

better candidates to be “independent”. An away game is independent when it is preceded and succeeded by a home game and hence is not in a road trip. A team will seek to play independent games close to their home site to minimize the travel to and from the opponent’s site. Figure 6.4 illustrates the costing for a tile of team PIT that includes travel to team PHI, NYM and MON. The left side of Figure 6.4 represents the tile completely broken, forcing a round-trip travel to each team in the tile. The right side represents maintaining the tile as a three team road-trip.



	ATL	NYM	PHI	MON	FLA	PIT
ATL	0	745	665	929	605	521
NYM	745	0	80	337	1090	315
PHI	665	80	0	380	1020	257
MON	929	337	380	0	1380	408
FLA	605	1090	1020	1380	0	1010
PIT	521	315	257	408	1010	0

Figure. 6.4. Example Tile Costing Diagram

Our hypothesis is that the use of tiles can be a predictor of road trips in the best solutions, and hence provide an initial solution leading to a high quality solution. We assert that the tiles with the highest cost of breakage will be found in the highest quality solutions. Tiles with low cost will be broken to complete a feasible schedule.

We provide evidence that the cost of a tile is a predictor by quantifying their presence within a known or target solution. A tile may be completely present in the target solution, if the tile exists as a road trip in that solution. A tile may be completely absent, if all three games in the tile are in separate road trips in the solution. There are other combinations of the tile appearing within the solution, including two of the tiles games existing on a common road trip, while the remaining game is in a different road trip.

We assign the following values to these comparisons to achieve an objective measure of a tile's presence in the target solution. For each game in our tile found within a road trip with another game of all tile, one point is assigned. Hence if the tile matches a road trip in the target solution, the tile is given a value of three. If two of the games in the tile are in a road trip within the target solution, and the remaining game is in a different road trip, the tile receives two points. For games in the tile that are independent in the target solution, one point is deducted for the tile. Hence, a tile where all three games are actually independent games in the target solution; the comparison value is negative three.

The point assignment is as follows:

1 point is assigned if the game exists in a road trip with another game in the tile

0 points are awarded if the game is in a road trip, but no other game in the tile is in that road trip

-1 points are assigned if the game appears in no road trip, hence an independent game.

Table 6.1 provides some common scenarios and the comparison value assigned.

Assigned Value	**Description**	**Tile**	**Target Solution Road Trips**
3	The tile exists in the solution, though games may be in a different order	1-2-3	3-2-1
2	Two games appear in the same road trip, while the remaining game is in another road trip	1-2-3	2-4-1 3-5-6
1	Two games appear in the same road trip, while the remaining game is independent. Games 1 and 2 are assigned 1, while game 3 is assigned -1.	1-2-3	2-4-1 7-5-6 3
0	All games appear in different road trips.	1-2-3	1-4-5 2-6-7 3-8-9
-1	No tile games are in same road trip, and one game is independent. Game 1 and Game 2 are assigned 0 points, while Game 3 is assigned -1	1-2-3	1-4-5 2-6-7 3
-2	Two games are independent while the third game is in a roadtrip.	1-2-3	1-4-6 2 3
-3	All tile games are independent games in the target solution. Each game is assigned a value of -1	1-2-3	1 2 3

Table 6.1 Examples of tile value calculations.

Using this evaluation we can measure the presence of our tiles against solutions of the best known approaches to date. The three approaches we used for comparison are presented in Table B. The first approach, described by Van Hentenryck & Vergados [115] is a parallel processing approach, carried out on clusters of dual-processor blade servers. They use a simulated annealing based Traveling Tournament algorithm, first proposed by Anagnostopoulos [6]. The second approach by Di Gaspero & Schaerf [34] uses tabu search. The final comparison is with Araujo et al. [7], using parallel processing. This approach is also a two phase approach, with a random initial solution construction, followed by a greedy search phase to reach a local minimum to produce a mirrored solution. The local search phase is iterated after a perbutation of the initial solution.

We choose instances with higher numbers of teams, since there are more tiles involved in these instances. Based upon our access to available results data, we look to compare against the NFL instances with 16, 22, and 26 teams. Note that the 16 and 22 team instances allow our tiling creation step to generate tiles all having three games, since $mod((n\text{-}1)/3) = 0$. For the 26 team instance, we will only compare the tiles with 3 games, ignoring the one tile of two teams, to enable a fair comparison across instances.

The detail tile-by-tile comparison is presented for the three instances in tables A1 through A3. The following tables provide a summary of comparison point value of tiles with costs in various ranges:

Cost Range	# of Tiles	Di Gaspero & Schaerf Score [34]	Van Hentenryck & Vergados Score [115]	Araujo et al. (GRASP) Score[7]	Di Gaspero & Schaerf Avg [34]	Van Hentenryck & Vergados Avg [115]	Araujo et al. (GRASP) Avg [7]	Avg. Point Value
9000-9999	2	6	6	4	3.00	3.00	2.00	2.67
8000-8999	4	11	11	10	2.75	2.75	2.50	2.67
7000-7999	4	11	11	10	2.75	2.75	2.50	2.67
6000-6999	7	17	19	18	2.43	2.71	2.57	2.57
5000-5999	4	11	12	12	2.75	3.00	3.00	2.92
4000-4999	6	13	16	14	2.17	2.67	2.33	2.39
3000-3999	9	19	15	12	2.11	1.67	1.33	1.70
2000-2999	17	17	17	17	1.00	1.00	1.00	1.00
1000-1999	14	26	23	21	1.86	1.64	1.50	1.67
<1000	13	2	7	-12	0.15	0.54	-0.92	-0.08

Table 6.2. NFL 16 team instance comparison.

Cost Range	# of Tiles	Di Gaspero & Schaerf Score [34]	Van Hentenryck & Vergados Score [115]	Di Gaspero & Schaerf Avg [34]	Van Hentenryck & Vergados Avg. [115]	Avg. Point Value
>10000	1	2	2	2.00	2.00	2.00
9000-9999	3	6	8	2.67	2.00	2.33
8000-8999	4	7	9	2.25	1.75	2.00
7000-7999	8	19	22	2.75	2.38	2.56
6000-6999	6	14	17	2.83	2.33	2.58
5000-5999	7	16	19	2.71	2.29	2.50
4000-4999	17	35	43	2.53	2.06	2.29
3000-3999	21	40	56	2.67	1.90	2.29
2000-2999	33	53	78	2.36	1.61	1.98
1000-1999	28	29	44	1.57	1.04	1.30
<1000	26	-18	26	1.00	-.069	0.15

Table 6.3. NFL 22 team instance comparison.

Cost Range	Num-ber of Tiles	Di Gaspero & Schaerf Score [34]	Van Hentenryck &Vergados Score [115]	Di Gaspero & Schaerf Avg [34]	Van Hentenryck & Vergados Avg. [115]	Avg. Point Value
>10000	1	1	2	1.00	2.00	1.50
9000-9999	4	7	5	1.75	1.25	1.50
8000-8999	4	10	7	2.50	1.75	2.13
7000-7999	8	15	22	1.88	2.75	2.31
6000-6999	9	18	18	2.00	2.00	2.00
5000-5999	12	22	25	1.83	2.08	1.96
4000-4999	15	27	33	1.80	2.20	2.00
3000-3999	39	73	89	1.87	2.28	2.08
2000-2999	37	66	76	1.78	2.05	1.92
1000-1999	36	43	52	1.19	1.44	1.32
<1000	16	-3	11	-0.19	0.69	0.25

Table 6.4. NFL 26 team instance comparison.

The results in Table 6.2 show that across the three best known solutions, the 21 tiles with costs higher than five thousand were found almost completely intact within these solutions, with an average score of 2.67. A score of 2.67 is achieved over a set of three tiles, by two tiles existing completely in the target solution, and the remaining tile having two of its three games sharing a road trip in the target solution. This scenario is almost a complete match of the generated tile and the target solution's road trips.

Conversely, tiles with low costs were found to be broken and their games often scheduled as independent games. Note the negative average total for the sixteen tiles costing under

one thousand. The games in these tiles were more often found as independent rather than belonging to any road trip.

Van Hentenryck & Vergados [115] have the best solution (235,930 total distance covered) of the three solutions. Note that their tile comparison scores are slightly higher than the other solutions. This is also true for Table 6.3 and Table 6.4, where the best solution has the highest comparison value. Hence, the best solutions are those that contain the higher cost tiles.

The tables show that the cost of our tiles are a predictor of whether that tile will exist in a high quality solution. Tiles with high cost appear, while tiles with low cost are broken and placed appropriately to reach a feasible solution.

6.3.3 Tile Placement

We select the tile with the highest cost. We place this tile on the first available space on the schedule without violating any constraints. We start with week one, and move forward checking all constraints for the three games represented in the tile. If the tile cannot be placed in week one, we rotate the tile, by changing the order of the first and last teams in the road trip. This rotation does not alter the distance associated with the tile. The tile is moved through the schedule week by week until a placement can be found.

The tile placement process is continued for each tile in cost priority order. When no more tiles can be placed, we break the remaining tiles into individual games. The games are

then placed in the schedule, starting from the first week of the schedule. We relax the maximum away games and home stand rules as well as the repeater constraint at this point to allow for placement of all games. When a game cannot be placed, given these relaxations, previous assignments are backtracked. If the backtracking of the individual games does not produce a feasible schedule, then the last placed tile is backtracked.

One noteworthy aspect of our approach is that tiles are never broken and then formed into new tiles, referred to as reconfiguration. This process involves breaking two tiles in their component blocks, and regrouping the blocks into two new tiles.

6.3.4 Local Search

The local search phase of our approach is designed to remove any hard constraint violations and improve the quality of the schedule. When it is possible to place all tiles, a near optimal solution based on the quality of our tile set can be generated. The breaking of the tiles results in a degradation of our initial solution. Hence, the games placed singularly are the cause of all hard constraint violations, and higher than best travel distances found using MST. Our local search seeks to reach the best local minimum of our initial solution after performing a variety of "swaps". Each swap moves a set of games within the schedule while maintaining, or improving, feasibility and the objective function of the solution.

We employ the following sets of swaps during our local search phase:

Home / Away Swap – We swap two games, involving the same two teams, where each team is home in one game and away in the other game. The swap is done between two weeks.

Round Swap – All games for two weeks are moved between the two weeks.

Partial Round Swap - The partial round swap described in [6], moves a set of connected games between two weeks. We begin this operation by selecting two games in different weeks. The teams in these games create our swap set. We then add teams playing these two teams, in either week, to the swap-set. The process continues until all teams, in both weeks, are in the swap set or its complement. We then move all games involving teams in the swap set between weeks. We can also choose to move all teams in the complement of the swap set between the weeks.

The local search phase is an iterative process to optimize the impact of the above swap actions. We look at each swap in the order above, and analyze the schedule improvement, if any, of each possible swap. The swap with the largest improvement is performed to reach a new schedule. This method is a steepest descent local search method. If no improvement can be found, we move to the next type of swap in the list above. The same process is used until no improvement can be found. After performing all possible swaps of the last type (partial round), we have reached a local minimum and stop the local search.

6.4. INSTANCES AND CURRENT RESULTS

The TTP problem is defined by Trick [109] along with several instances of the problem and solution results submitted by various researchers. The initial set of instances was

based upon the cities that house a team for the National League (NL) of Major League Baseball in the United States. An instance was developed for all even numbers from 4 to the complete set of 16 teams in the NL. A distance matrix was provided to provide the distances between all pairs of cities. The instance is noted by "NL," followed by the number of teams in the instance. Table 6.5 shows the best results as of this writing for the top solutions by instance. The table provides the approach, the supporting reference and the result for each instance.

Approach	***NL 8***	***NL10***	***NL12***	***NL14***	***NL16***
Anagnostopoulos et al. [6]			111248	189766	273802
Cardemil [22]	40416	66037	118955	205894	308413
Easton et al. [42]	**39721**				
Langford [66]		**59436**		190056	272,902
Van Hentenryck et al.[115]		59583	**110729**	**188728**	**261687**
Shen & Zhang [105]	39947	61608	119012		293175

Table 6.5. Best NL instance results.

Bold notes best solution

A second set of instances were then developed based upon the National Football League (NFL), playing American football in the United States. This league has 32 teams, and again instances were developed for instances of even numbered teams. Since the NL problem is similar to NFL instances, only NFL instances from 16 teams to 32 teams are tracked by Trick [109]. Table 6.6 and Table 6.7 provide the best results to date for the NFL instances.

Approach	***NFL 16***	***NFL18***	***NFL20***	***NFL22***	***NFL24***
Araujo, Urrutia, & Ribeiro [7]					
Di Gaspero & Schaerf [34]	238581	439626	352947	439626	499017
Ribeiro & Urrutia [99]				418086	467135
Langford [66]	235930	296638	346324	412812	490528
Van Hentenryck & Vergados[115]	**231483**	**282258**	**332041**	**402534**	**463657**

Table 6.6. Best NFL instance results NFL16 through NFL24.

Bold notes best solution.

Approach	***NFL 26***	***NFL28***	***NFL30***	***NFL32***
Araujo et al. [7]		**609788**	**739,697**	**914,620**
Di Gaspero & Schaerf [34]	590,497	681,197	847,011	1,020,966
Langford [66]	551,033	669,320		
Ribeiro & Urrutia [99]	554,670	618,801	740,458	924,559
Van Hentenryck & Vergados[115]	**536,792**			

Table 6.7. Best NFL instance results NFL26 through NFL32.

Bold notes best solution.

Another set of widely studied instances related to the TTP problem, but not a particular set of cities is the constant distance matrix. In these sets of instances, all distances between cities are noted to be one. The instances studied are those from 4 teams in the league up to 24 teams. An instance is named by the word “Con” and the number of teams in the league for that instance. We show the results of the best approaches in table 6.8.

Approach	*Con8*	*Con10*	*Con12*	*Con14*	*Con16*	*Con18*	*Con20*	*Con22*	*Con24*
Fujiwara [45]								**626**	
Rasmussen & Trick [96]					**327**				
Di Gaspero& Schaerf [34]		**124**	**181**	**252**	329	418	521	632	757
Ribeiro &Urrutia [99]	**80**	130	192	256	342	434	526		
Van Hentenryck &Vergados [115]						**417**	**520**	628	**749**

Table 6.8. Best TTP Constant Distance Results.

Bold notes best solution.

Trick[109] presents other leagues and variations on the distance matrix. The instances above serve as the best set to determine the effectiveness of the approach within the real world environment. This thesis focuses on instances of the above set of problems of 16 teams or more, where a variety of approaches are available for comparison. The 16 team number is the start of the NFL instances and is the only overlapping instance between the NL and NFL instances. Also resources used in most approaches do not substantially grow until the instance contains at least 12 teams.

We seek to develop a solution approach for leagues with a large number of teams that will produce a high quality solution in a short period of time. Instances with 16 teams or more provide the challenge necessary to test this approach.

6.5. OUR RESULTS

Our two phase approach of tiling and local search enables us to produce a high quality solution with minimal resources. We only use a small fraction of the resources consumed by the best known approaches. Our platform is a 2.13 Ghz Dell laptop, using a .NET software application.

The following tables compare our solutions to a variety of TTP instances. We choose instances with a higher number of teams and the constrained constant distance instances of the problem, for comparison. We also only consider approaches with multiple results across instances and supporting research documentation. The instances with fewer teams have been addressed by a variety of algorithms, whereas instances with 16 teams or more have been addressed successfully by approaches we consider below.

Instance	**Tiling Results**	**Van Hentenryck and Vergados Results [115]**	**Di Gaspero and Schaerf Results [34]**	**Araujo et al. (GRASP) Results [7]**	**Average % Difference from Tiling**
NL 16	317,764	**267,194**	279,465	285,614	14.54%
NFL 16	266,231	**235,930**	238,581	248,818	12.21%
NFL 18	339,822	**296,638**	N/A	299,134	14.08%
NFL 20	406,463	**332041**	352947	359,748	16.71%
NFL 22	482,374	**412,812**	439,626	418,086	13.90%
NFL 24	544,354	**463,657**	499017	465,491	14.34%
CON16	354	N/A	**328**	342	5.67%
CON18	466	N/A	**418**	432	9.64%
CON20	568	520	**521**	522	9.02%

*Table 6.9 Results Comparison (best solutions shown in **bold**).*

Instance	Tiling Time	Van Hentenryck and Vergados Time – Best [115]	Di Gaspero and Schaerf Time – [34]
NL 16	38	1,815	51,022.4
NFL 16	35	2,220	N/A
NFL 18	105	3,120	N/A
NFL 20	135	6,750	N/A
NFL 22	150	8,100	N/A
NFL 24	320	5,490	N/A
CON16	18	N/A	19,665
CON18	22	N/A	33,979
CON20	23	N/A	46,579

Table 6.10: Time Comparison in seconds.

Table 6.9 compares the results and resources of our tiling approach with the best-known approaches. Our tiling approach comes within 10-16% of the objective function for the TTP for all approaches. The tiling approach produces slightly better results where the number of games to be played by each team is divisible by three. This situation allows all games for a team to be placed in three-team tiles. Hence our 16 and 22 team instances are slightly better in comparison with these best-known approaches.

For the constrained instances, our approach is even closer than the higher NFL instances. The construction and costing of the tiles is not important, since all distances are constant. Hence, each tile for a team has the same cost, when the tile is broken. Our 16 team instance, dictating 5 tiles of three teams per tile, provides our best result. One factor in this result is that the games for each team can be constructed into a set of tiles with three games each. This enables more tiles to be placed in the initial solution, increasing the quality of the initial solution.

We use only a few seconds compared to the substantive time used in the other approaches. Table 6.10 compares our times with the published times of the approaches with comparable metrics.

The key difference between our tiling approach, and the other approaches, is the initial solution. Our initial solution is built based upon tiles, which are high quality partial solutions. Hence our approach saves the time used by others in the local search to develop a high quality initial solution. We follow our tiling phase with an efficient greedy approach to quickly develop a solution relatively close to the best known solutions.

The most illuminating result is the NL16 figure for Araujo et al. in Table 6.9. The result of 285,614 was achieved by running GRASP algorithm [7] for five days. This algorithm is similar to our local search, because of its greedy nature. Both algorithms explore all possible swaps of a given category, and perform the swap with the greatest improvement. When no improvements can be made, the local minimum is reached. Since the local search phases are similar, the key difference between the approaches is the construction of the initial solution. After each local minimum is reached, random swaps are made to create a new solution. Hence large numbers of initial solutions are created over the five day processing period. The best of these solutions led to only an 8.5% improvement of creating one initial solution solution through tiling.

6.6 SCALABILITY AND FLEXIBILITY

The result comparison in the previous section demonstrates how the tiling approach provides a solution close to the best published solutions to date. These solutions, however, take advantage of tremendous amounts of resources and problem assumptions that would prohibit their use in a large number of real world problems. For example, the use of mirroring would not be applicable in leagues with an unbalanced schedule. Other approaches use simplistic round-robin tournament generators which are not applicable in a wide variety of circumstances. We demonstrate the scalability and flexibility of our approach by applying it to the actual MLB schedule. The TTP problem has its roots in this league's schedule. For simplicity reasons, several key characteristics of the true MLB schedule were not included in the TTP problem definition. The two key characteristics, mentioned above, are unbalanced schedules and teams playing an unequal number of home and away games against the same opponent. These constraints eliminate the mirroring approach used above, and complicate the swapping steps of these approaches.

The tiling approach enables us to address the true MLB schedule effectively for several reasons. First, tiling does not depend upon a tournament construction process common in round-robin tournament approaches. The tiling approach can handle any set of matchups provided to the scheduler. Tiling also does not have a model with dependencies on home and away games, and does not perform any mirroring of partial schedules.

We look to test our approach on the true MLB schedule by first creating smaller representative instances. We create these instances with unbalanced schedules for 2, 4

and 6 divisions, where each division is an actual MLB division. We follow these instances with the true set of matchups necessary for the MLB schedule of 2010. The complete instance definitions, including matchups and distance matrices, can be found on the TTP problem definition web site [109].

Table 6.11 describes these instances in terms of the number of round-robin schedules involves between divisions. A complete definition of the instance, including distance matrices, matchups, tiles and solutions are provided in tables A4 through A43.

Instance	**Teams**	**General Description**
DIV1	1 division of 6 teams	Each team plays every other team 2 or 3 times at home or away.
DIV2	2 divisions of 5 teams each	Each team plays a double round-robin within its divison and a single round-robin against the other division.
DIV4	4 divisions of 5 teams each	Each team plays a double round-robin within its divison and a single round-robin against two other divisions.
MLB	Actual MLB with 6 divisions with 4-6 teams each.	Generally each team play 3 single-round robin within its division, a single round-robin with other divisions within its league, and a small set of games against selected opponents in the other league.

Table 6.11: New TTP instance definition.

We modified one constraint of the TTP to state that the same team cannot be in the same away trip or home stand. This scenario is not possible in the current TTP instances since each team is only a road opponent once in the season.

Table 6.12 shows the results of our tiling and local search approach for these instances.

Instance	**Tiling Time** (secs)	**Results**
DIV1	63	54,801
DIV2	84	168,429
DIV4	135	381,066
MLB	127	959,944 (actual MLB: 1,000,588)

Table 6.12: New TTP instance results.

Based upon similar TTP instances, our results look to be in close range of existing approaches. Our MLB result was actually better than the true MLB 2010 schedule, however the actual MLB schedule has several constraints (e.g., black-out dates, specific times for inter-league play, and distribution of key team match-ups across the season) not included in our modified TTP definition. However, the comparison serves to show that our solution is a high quality solution.

6.7 CONCLUSION

Our approach demonstrates how tiling can be used effectively to produce high quality solutions for the TTP problem. Our conjecture that road trips that appear in the best solutions are also present in the tiles we create. Tiles with high cost are bound to appear, while tiles with low cost will be broken and scheduled as independent games. We demontrated our conjecture through a graph theory based algorithm. For other timetabling problems, the conjecture may be derived from historical results or business

knowledge. The tiling represents a method of using pre-existing knowledge to develop an initial solution that can quickly produce a high quality result.

The best known approaches use resource intensive local search techniques to perform swapping, which ultimately results in building the high cost tiles in our solution. We show how starting with tiles representing known desired road trips, can reduce these computational resources.

The ability to produce a fast, high quality solution is critical since the scheduling process itself is an iterative process. Users of the schedule look at initial drafts and modify constraints and objectives accordingly. Trick in [113] describes how the production of the MLB schedule involves many rounds of schedule production and analysis. Also, new constraints about site availability may arise during the scheduling production itself. In addition, not all organizations are professional sports leagues with tremendous resources to pay for expert consultation necessary to implement the best known algorithms cited in this thesis. Our frugal use of resources is required for organizations on a tight timeline and budget.

Beyond simply speed and performance, we show how our tiles can embed business knowledge about the problem. For example, Trick [113] mentions how the real MLB scheduling process involves a specialist with logistical knowledge about the traveling between the various cities. When schedules are produced, this specialist notes some soft constraints about the sequence of cities. For instance, it may be that when traveling from

Milwaukee to Houston, there is no direct flight and a long layover between connecting flights. Hence, this sequence in the schedule is allowable but not desired. Our tiling approach would accommodate these types of requests. Initially all tiles would not violate this soft constraint. After the tiling phase, the soft constraint would only be violated during the swapping phase, if necessary, to achieve a feasible solution. The other approaches we have seen would need to modify the objective function. The objective function for these approaches would need to trade distance for adherence to the soft constraint, in order to drive the constraint processing engine.

We also introduce new instances of the TTP that more accurately mirror the true MLB scheduling challenge. These instances provide a more real world problem definition than a simplistic round-robin tournament. Our tiling approach can handle the complexities of the true MLB requirements, and provide support for the iterative scheduling process involved in producing a professional sports schedule.

The next chapter introduces a more prevalent sports scheduling problem in the timetabling community. The chapter defines the problem and demonstrates how tiling was applied successfully to one real world instance of the problem.

CHAPTER 7 – YOUTH SPORTS LEAGUES

The scheduling of youth sports leagues is the most common sports scheduling problem found in the world today. Unlike professional sports scheduling challenges, youth sports leagues share venues, play unbalanced (playing opponents an unequal number of times) schedules, and must coordinate with the official scheduling process. Youth sports league typically contain hundreds of teams which must be scheduled simultaneously. We define the youth sport leagues problem and present a tiling approach to one instance.

7.1 INTRODUCTION

Youth sports are administered by governing bodies that determine sportsmanship rules, promote the sport, and organize youth participation. Organizations within these bodies may be towns, high schools, religious groups and sports clubs with international affiliations (e.g., FIFA -Federation International de Football Federation). Each of these organizations sponsor teams in leagues, and provide a venue or fixture. For example, youth leagues in the United States include: junior soccer leagues, Little League baseball, inter-scholastic high school basketball [73] , and the Catholic Youth Organization (CYO) [87]. Youth sports leagues are played worldwide. For example, Little League Baseball is played in 72 countries worldwide within 7,170 leagues, comprising over 2 million players [72].

The Youth Sports league community uses the following terminology:

- *Capacity*: The number of games that can be played at a given venue on any day. The capacity may vary by venue.

- *Division:* A set of teams where every team in the division plays against every other team a set number of times. Subsets of these teams may play one additional game amongst them to reach the required number of games.

- *Game:* A sports contest between two teams.

- *League:* A set of divisions, comprised of teams, which share a common set of venues.

- *Organization:* An entity that maintains a venue and sponsors teams across many divisions.

- *Round-robin tournament*: A round-robin tournament involves *n* teams playing *n*-1 games. Each game is played against a different opponent in the tournament. Also, every team plays a game in each round.

- *Slot*: A time period during a given day to be used to play a game at a venue. The YSL schedules a fixed number of slots per day at each venue, depending upon the venue's daily capacity. The actual clock time of the slot is not considered in the problem.

- *Unbalanced Schedule*: A schedule where a team will play one opponent more often than another opponent. The YSL requires unbalanced schedules in divisions where the required number of games in a season is not a multiple of the number of n 1, for a division.

- *Venue:* A physical location for a game to played.

A Youth Sports League (YSL) consists of divisions that are sets of teams grouped by age, gender, and/or level of play. The number of teams in a division can vary, ranging from 4 to 20 teams. Each individual participant registers with the league to play the same number of games, regardless of division. The schedule for a division is often a round robin tournament followed by additional games against selected opponents from the division in order to meet each team's required number of games. This type of schedule is referred to as "unbalanced" since a team may play one opponent once more than another. The sharing of the organization's venue by its sponsored teams creates a dependency between the division schedules. Two teams from the same organization, possibly from

different divisions, cannot host a game at the same time. Hence, the administrator must consider the schedule of all divisions, when creating the master schedule.

The problem of scheduling youth sports leagues differs from scheduling professional sports league. This latter problem has been widely studied in the scheduling literature and has recently been surveyed by Kendall et al. [59]. McAloon et. al. [76] looked at semi-professional tournaments, Wright [120] , Bean [11] addressed professional baseketball, Croce & Oliveri [29] , Bartsch [10] , Schreuder [104] examined professional soccer leagues, Cain [21] major league baseball, Armstrong and Willis [8], Willis & Terrill [119], Wright [121] solving professional cricket leagues, and Fleurent & Ferland [44] studied the professional hockey in the United States. Virtually all professional sports have been a subject in research literature.

There are significant differences and challenges to the two types of problems. Professional sports usually involve a balanced schedule, with guaranteed availability of the venue. Youth sports leagues play unbalanced schedules, and teams from all divisions must share a venue. Professional sports teams' venues typically only host one game in a day whereas a YSL venue can support several games a day, This venue sharing creates a schedule dependency among all divisions. A professional league with 4 divisions comprising 12 teams, can be viewed as four separate and distinct round-robin tournaments. The same league structure in the YSL must be viewed as one schedule with 48 teams playing an unbalanced schedule. Real world instances of youth sports can

include 400 divisions encompassing 3,500 teams, as is the case for the Long Island CYO youth basketball league.

In the following sections, we present the background of the challenges facing a multitude of youth sports league worldwide. We then present a problem definition that includes common constraints and a representative objective function. This problem definition can be used by researchers to develop techniques to address the problem faced by these organizations. We conclude by demonstrating the effectiveness of our proposed tiling methodology in solving the youth sports league problem for a league with several thousand teams.

7.2. BACKGROUND

7.2.1 League Formation

Youth sports leagues are participation oriented. In contrast, the sole focus for professional sports leagues is on competition. The league formation process begins with a registration period where interested players request entry onto a team. The organization then assigns players to specific teams. Teams formed from these registrations are typically done according to age, gender and ability. In some instances, registration of a previously formed team may occur. In either case, the teams are then assigned to divisions. A division is defined as a set of teams to be involved in one or more round-robin tournaments. While some leagues include inter-divisional play, the YSL problem will only consider fixtures within a division. Soon after division formation, the league is created and the season games begin.

The importance of discussing the registration process is two-fold. Firstly, since all teams are accepted, there is a wide variation in the number of teams within a division. However, the league requires that all teams must play the same number of games. The scheduler must schedule one or more round-robin tournaments. In order to equalize the number of games across divisions, a partial round-robin tournament may be scheduled, where only certain rounds of the tournament are considered. For example a division of 6 teams that must play 12 games, would play a double round-robin tournament and two rounds of a partial round-robin tournament. The concept of teams playing an unequal number of games against each opponent is more acceptable for youth sports leagues than for professional sports leagues for two reasons. Firstly, competition is not the focus of the league, but rather player participation and development. Many youth sports leagues have rules to move the focus from winning to participation. For instance in some leagues, each player must participate in the game for a minimum amount of time. Secondly the importance of the registration and league formation process is carried out in a short-time frame. DiNunzio & Kruk [39] describe the intense pressure and speed of youth soccer tournament scheduling. Their work focuses on developing a tournament schedule when the number of teams in the tournament is not known until one hour before the first game. The YSL problem also requires the scheduler to quickly devise the schedule and distribute the schedule to team coaches for approval. This process becomes iterative as coaches identify more team and venue constraints after seeing the schedule. It is crucial that the turnaround time for the league schedule be a matter of days or hours to support this process.

7.2.2 Instances of the YSL in the Real World

Our proposed YSL scheduling problem is not aimed at one organization, or even one sport. We look to address the problem of youth sports scheduling across countries, sports, gender and age groups. The following is a sample list of organizations and leagues that would be representative of the YSL scheduling problem described in this thesis:

- **Catholic Youth Organizations of America** (http://www.cyonetwork.org).
- **Catholic Youth Organization of Nassau-Suffolk County**, (http://www.cyons.org).
- **England BasketBall** – Fixtures and Results (http://www.englandbasketball.co.uk/fixtures/fixtures-results.aspx).
- **Football mitoo** (http://www.football.mitoo.com/ListOfLeagues.cfm)
- **Long Island Junior Soccer League** (http://www.lijsoccer.com/leagueplay.html).
- **Suffolk County Athletics** (http://www.sectionxi.org).
- **Westchester-Putnam County CYO Basketball.** (http://www.wpcyo.org).

While each organization or league may have its own particular constraints, we believe approaches developed for our YSL problem would be applicable to the vast majority of youth sports represented in above.

7.3 PROBLEM DEFINITION

The YSL involves scheduling multiple divisions, each containing teams sponsored by a common set of member organizations, across a set of venues. All games are intra-divisional. A season consists of a set number of games to be played by all teams in the league, regardless of the number of teams in a particular division. If the number of games in a season is 12, then teams in a five team division and an eight team division will all play exactly 12 games. Some divisions are required to play an unbalanced round-robin tournament, where a team will play various opponents a different number of times.

Each organization makes a venue available to the league. The venue has a daily capacity or number of games that can be played on one day. A league may allow games to be played on more than one day of the week, particularly weekends. A season consists of a set of consecutive weeks, to be played on a given number of days per week. A venue's season capacity is the number of games per day that can be hosted by the venue, multiplied by the available days in the season. Divisions also have an associated referee level, indicating the minimum level of the referee's certification. This level is referred to as the referee category. A referee must have the proper level of certification to officiate the game. Referees may choose to officiate games at a lower level than their certification based upon their personal preference. This preference may stem from a wish to provide more on-field rule instruction to the younger age groups. However, in this instance the referee would be assigned a lower level category for the season. Each referee is assigned a category based upon their certification and personal requests. We make the assumption that each referee has only one category, which is the more common scenario. Generally a referee is assigned two games at a venue, on a given day. The referee prefers to have the games follow each other to minimize his or her time at the venue. Hence it is desirable to schedule games, requiring the same referee level in succession. A venue's daily schedule should have an even number of games for each referee category to support this concept. This will enable the scheduler to schedule two consecutive games for a referee. A referee is permitted to officiate games consecutively from different divisions, if the referee's category allows it.

The schedules for the individual divisions are combined into a master schedule. The master schedule must address all venue sharing and referee assignment constraints.

A master schedule is to be created for all divisions such that these hard constraints cannot be violated:

H1: Teams must play exactly *G* games within *W* weeks, where *G* is the league wide number of games per team, and *W* is the number of weeks of play during the season.

H2: Each division will play a multiple round-robin schedule. For divisions requiring unbalanced schedules, teams may play an opponent only one additional time more than other opponents.

H3: Teams can play multiple games in a week, but only one game per day.

H4: Each venue has a maximum number of games C that can be played at the venue on one day. This value differs by venue. Each team has an associated home venue.

Our research recognizes that one problem definition could not encompass all the goals and objectives of the thousands of existing youth sports leagues. We look to define a problem that contains a goal for each of the major perspectives of a master schedule within a YSL scheduling problem. Figure 7.1 shows different perspectives of the same league schedule.

Team's Calendar View

Figure 7.1– Team calendar view perspective.

Date	Team's Game Sequence
5/7	H - T6 (1pm)
5/15	A - T8 (4pm)
5/21	A - T4 (3pm)
5/22	H - T8 (1pm)
5/30	H - T2 (2pm)

H denotes Home game
A denotes Away game

Figure 7.2– Team season view.

Venue's Daily Schedule

	Time
D1- T2 vs. T3	1pm
no game	2pm
D2- T1 vs. T6	3pm
D2- T2 vs. T5	4pm
D3- T5 vs. T7	5pm

Dn is Division Number
Tn is Team Number

Figure 7.3– Daily venue schedule.

Figure 7.1 considers how the games will fall for a given team on a calendar. The scheduling objectives from this perspective include team availability, maximizing or minimizing playing games on a particular day of the week, minimizing games on holiday weekends, and standardizing lag time between games. For our YSL scheduling problem definition, we choose minimizing playing multiple games during a week as a factor in the objective function. For large youth sports leagues with tight venue capacity constraints, it is often difficult to avoid multiple games in a weekend.

Figure 7.2 analyzes the sequence of opponents for a team and their travel requirements. Objectives, based upon this perspective, include back to back games with the same opponent, timing historical rivalries at a certain point in the season, and the sequencing of home and away games. Our objective function seeks to have teams alternate home and away games throughout the season.

Figure 7.3 considers the games for a particular venue on a given day. In addition to the obvious slot availability constraint, schedulers of youth sports leagues frequently have several objectives for the set of games at a venue. The scheduler is keenly aware of the referee assignment task, and will often look to schedule consecutive games requiring the same level of officiating. This will enable a referee to officiate two games. The scheduler may also look to schedule teams from the same organization as the away team, so there can be car-pooling and other travel arrangement sharing. Teams involving younger players may be scheduled earlier in the day, since the probability of extended games

extending the schedule is reduced. Our YSL scheduling problem's objective function rewards the scheduler for consecutive games of the same referee level.

The quality of the schedule is evaluated by calculating the number of soft constraint violations within the schedule. Schedules with fewer soft constraint violations are considered of higher quality. The following section lists the soft constraint violations and their weights in brackets.

S1[4]: A team playing more than one game in a week. Teams may play only one game per day, but are permitted to play on multiple days. For example, a team having games on both weekend days would be assigned four soft constraint violations.

S2[2]: A venue hosting a set of games where a referee is not able to officiate back to back games. A game that cannot be paired with another of a division within the same referee category would count as a soft constraint violation.

S3[1]: Teams prefer to alternate home and away games. A team playing home (or away) games consecutively (usually referred to a break) would count as one soft constraint violation.

The YSL scheduling problem solution must be a feasible solution satisfying all the hard constraints. The objective function of the YSL scheduling problem is to minimize

the number of weighted soft constraints above. Regin [97] shows that for a round-robin tournament, it is not possible to eliminate all breaks. Hence, the third soft constraint renders a schedule of no soft constraint violations for the YSL scheduling problem impossible.

A formal model of the YSL problem is given presented below:

Inputs

V: set of venues where games are played
D: set of divisions, each with a varying number of teams
T: set of teams for competition
R: set of categories indicating the certification level of referees
Y: set of days of the week eligible for game play
S: set of slots within the day
D_{td}: 1 if team t is a member of division d.
0 otherwise
V_{tv}: 1 if team t plays its home games at venue v
0 otherwise
R_{dc}: 1 if division d has referee category c
Y_{yp}: 1 if day y includes period p
G: The number of games all teams are required to play
W: The number of weeks, in elapsed time, of the season
Y: The number of days per week all venues can be used
C_v: The daily capacity of games that can be played at venue v

Decision Variables

H_{tws} 1 if team t is host team for a game during week w in period s.
0 otherwise.
A_{iws} 1 if team t is visiting team for a game during week w in slot s.
0 otherwise.
P_{ijws} 1 if team i plays team j in week w in slot s.
0 otherwise.
V_{vws} 1 if a game is scheduled for a venue in week w in period s.
0 otherwise.
R_{rvws} 1 if a game is scheduled for category r at venue v during week w in period s.
0 otherwise.

Objective Function: Minimize F(x) = 4*S1 + 2*S2 + S3
where

S1: $(\sum_{s=1}^{S} H_{tws} + A_{tws}) - 1 \; \forall \{t \in T, w = 1 \ldots W\}$

S2: $R_{vrws} != R_{vrw(s-1)}$ and $R_{vrws} != R_{vrw(s+1)}$

$\forall\{v \in V, r \in D, w = 1 \ldots W, s = 2 \ldots S-1)$

S3: $\sum_{s=1}^{S} H_{tws} \geq 1$ and $\sum_{s=1}^{S} H_{t(w+1)s} \geq 1 \ \forall\{t \in T, w = 1 \ldots W-1\}$

$\sum_{s=1}^{S} A_{tws} \geq 1$ and $\sum_{s=1}^{S} A_{t(w+1)s} \geq 1 \ \forall\{t \in T, w = 1 \ldots W-1\}$

Subject to:

H1: $\sum_{w=1}^{W} \sum_{s=1}^{S} H_{tws} + A_{tws} = G \ \forall t \in T$

H2: $\sum_{w=1}^{W} \sum_{s=1}^{S} (H_{tws} + A_{tws}) * Y_{ys} \leq 1 \ \forall t \in T, \forall y \in Y$

H3:

$\forall\{\text{divisions in D , each triplet of teams t, u, v from that division}\}$

$$\left| \sum_{w=1}^{W} \sum_{s=1}^{S} P_{tuws} - \sum_{w=1}^{W} \sum_{s=1}^{S} P_{tvws} \right| \leq 1$$

H4: $\sum_{s=1}^{Sn} V_{vws} \leq C_v \ \forall\{v \in V, w = 1 \ldots W\}$

7.4 EXAMPLE YSL INSTANCE

Our example problem instance involves three towns, referred to as A, B, C, each sponsoring one or more boys and girls teams, within the league. Table A shows the input parameters necessary for the YSL scheduling problem in our sample league. The boys division contains two teams from town A, and one team from B and C. The girls division has three teams from B, two teams from A and one team from C. Each town maintains one venue with a capacity of two games per day, on two days each week. The season should contain 8 games per team, over a 7 week season. Both divisions have the same referee category. Note that the boys division is a double round-robin tournament followed by additional games causing an unbalanced schedule. The girls division has a single

round robin tournament followed by additional games also resulting in an unbalanced schedule. The two divisions share a venue provided by each town.

Table 7.4 provides the set of inputs needed to define a YSL scheduling problem instance. Each row contains an input parameter, or a set of information about the league's structure.

Input Parameter	Value	Definition
V:	{x,y,z}	set of venues where games are played
D:	{b,g}	set of divisions, each with a varying number of teams
T:	{0-A1, 1-A2, 2-A3, 3-A4, 4-B1,5-B2,6-B3,7-B4, 8-C1,9-C2 }	A set of teams for competition. The team number is shown with the team symbol used in Table B and Table C.
R:	{1}	set of categories indicating the certification level of referees
Y:	{6,7} (this represents Saturday, Sunday)	set of days of the week eligible for game play
S:	{1,2,3,4}	set of slots within the day
D_{td}	D_{0b},D_{1b},D_{2g},D_{3g}, D_{4b},D_{5b},D_{6g},D_{7g}, D_{8b},D_{9g} = 1	1 if team t, found in T, is a member of division d found in D. 0 otherwise
V_{tv}:	V_{0x},V_{1x},V_{2x},V_{3x}, V_{4y},V_{5y},V_{6y},V_{7y}, V_{8z},V_{9z} = 1	1 if team t plays its home games at venue v found in V. 0 otherwise
R_{dc}:	R_{b1},R_{g1} = 1	1 if division d has referee category c
Y_{yp}:	R_{61},R_{62},R_{71},R_{72} = 1	1 if day y includes period p
G:	8	The number of games all teams are required to play
W:	7	The number of weeks, in elapsed time, of the season
Y:	2	The number of days per week all venues can be used
C_v:	C_x = 2, C_y=2, C_z=2	The daily capacity of games that can be played at venue v

Table 7.4 – Sample problem input parameters and data.

Table 7.5 provides the rounds necessary to schedule the master schedule for this youth sports league. A team is referred to by its town (A, B or C), followed by the sequence

number of the team within the town. The team reference of "A2" would indicate this team is the second team sponsored by town A. For the Boys division, the first round shown (A1-B1, A2-C2) has been placed in week 2 of the master schedule. Each team plays an unbalanced schedule. Team A1, in the boys division, plays B1 and C1 three times while only playing A2, twice. Each box of the master schedule represents a venue's games for a day, whose maximum is two in our example instance. In weeks where only one day is used by a venue, the choice of the day is not consequential.

The soft constraint violation calculations are shown at the bottom of Table 7.6. Since the number of games in a season is more than the number of elapsed weeks, it is guaranteed that soft constraint violations will be earned for each team due to the soft constraint of playing multiple games in a week. Since both divisions have the same referee category, venues hosting an even number of games, will not incur a soft constraint violation for nonconsecutive games by referee category. For example, in week 2, A2-C1 was scheduled on the same day as A4-C2 to avoid having an odd number of games on each day.

Boys	**A1-B1** **A2-C1** **-2-**	**A1-A2** **B1-C1** **-4-**	**C1-A1** **B1-A2** **-1-**	**B1-A1** **C1-A2** **-7-**	**A2-A1** **C1-B1** **-7-**	**A1-C1** **A2-B1** **-3-**	**A1-B1** **A2-C1** **-5-**	**B1-A2** **C1-A1** **-6-**
Girls	*A3-A4* *B3-B4* *B2-C2* *-7-*	*B4-A3* *B2-B3* *C2-A4* *-2-*	*A3-B3* *A4-B2* *C2-B4* *-3-*	*B2-A3* *A4-B4* *B3-C2* *-6-*	*B4-B2* *B3-A4* *A3-C2* *-1-*	*A4-A3* *B4-B3* *C2-B2* *-4-*	*A3-B4* *B3-B2* *A4-C2* *-2-*	*B4-A3* *B2-A4* *C2-B3* *-5-*

Table 7.5 – Tiles (Rounds) for the example league divisions.

Venue /Week	1		2		3		4		5		6		7	
A	C1-A1 B1-A2	*B3-A4*	*B4-A3 C2-A4*				A1-A2 *A4-A3*		*B4-A3 B2-A4*		B1-A2 C1-A1	*B2-A3*	B1-A1 C1-A2	A2-A1 *A3-A4*
B	*B4-B2*		A1-B1 *B2-B3*	*A3-B4 B3-B2*	*A3-B3 A4-B2*	A2-B1 *C2-B4*	*B4-B3 C2-B2*		A1-B1 *C2-B3*			*A4-B4*	C1-B1	*B3-B4*
C		*A3-C2*		A2-C1 *A4-C2*	A1-C1		B1-C1		A2-C1		*B3-C2*			*B2-C2*

Penalty Points Total = 95 points

1) **Multiple Games in a week: all Boys Teams week 7, all Girls Teams week 2 10 occurrences (40 points)**
2) **Days with odd number of Games in Referee Category: 12 days (24 points)**
3) **Playing Consecutive Home or Away games A1(4),A2(2),A3(3),A4(3) B1(1),B2(4),B3(3),B4(4), C1(4),C2(3) = (31 points)**

Table 7.6 – Sample Problem Scheduling Grid.

7.5 RELATED WORK

A great deal of research in sports scheduling has been aimed at professional sports leagues. Kendall et al. [59] provides a broad survey of various sports leagues and their scheduling challenges. Leagues are often structured as single or double round-robin tournaments. The scheduling challenges revolve around the quality of the schedule in order to achieve certain objectives. Norhona et al. [84], Ribeiro & Urrutia [99], consider fairness in South American football leagues, Kendall [57] seeks to minimize travel distances in the English Football league, and Rasmussen [95], and Goossens & Spieksma[48] consider various venue availability constraints. The instances in these problems can be reduced to tournament schedules of 20 to 24 teams, with consideration for some unique constraints and minimizing travel distance. Only Kendall [57] introduces a relationship between different divisions due to the pairing requirement (sets of venues that cannot host a game on the same day). In all these professional sports problems, a venue can be used for only one game per day.

The YSL scheduling problem is most closely associated with the Travelling Tournament Problem (TTP) described by Easton et al. [42]. The TTP and YSL share constraints, with the following exceptions:

1. The YSL may schedule multiples games per week, on separate days.
2. A YSL team is not required to play in every week.
3. Teams share home site venues, which support several games per day.
4. A YSL division may play an unbalanced tournament schedule.

These differences have a profound effect on the instance size, within real-world applications. The TTP instances documented on the problem definition web site [109] involve a maximum of 32 teams, albeit there is no real-world example for that size of a tournament. The YSL has common occurrences of leagues with 50 divisions, comprising over 500 teams. The CYO of Westchester-Putnam, near New York City, has 68 divisions, comprising 582 teams sponsored by 58 parishes.

7.6 . PROPOSED APPROACH

Our approach is a two phase approach based upon “tiling”. Moody et. al [81] introduces this approach in addressing the Traveling Tournament Problem (TTP). In the YSL scheduling problem, each tile represents a round of games for a division. A YSL instances would require a total number of tiles equaling the number of divisions multiplied by the games per season. Each tile is one round of a tournament for a division. The tiles are placed in the master schedule in a greedy fashion. Each tile is placed in a week that results in the least number of soft constraint violations being added to the objective function. Tiling continues until a tile cannot be placed without causing a hard-constraint violation. At this point, the venue capacity constraint is relaxed, and the

remaining tiles are placed in the schedule. These final placements continue to use greedy approach to minimize the total number of soft constraint violations.

Once all tiles are placed, we analyze each game's contribution to the objective function. Games violating the venue capacity hard constraint are temporarily assigned an artificially large soft constraint violation weight to prioritize them, within our local search phase. The schedule is now ready for the second phase, a local search. This phase consists of a set of swaps to reduce the current soft constraint violations of the schedule. The following sections discuss the steps of these two phases in more detail.

7.6.1 Tiling Phase

For each division, we use a single round-robin tournament to produce n-1 rounds of opponent pairings, where n is the number of teams within a given division. We use one of a variety of single round-robin tournament method generators discussed in [38]. The YSL scheduling problem requires a number of teams, g, for each team to play. Hence, g / (n-1) is the number of round-robin tournaments needed for each division, The YSL scheduling problem will frequently have an unbalanced schedule. This occurs when g / (n-1) is not an integer. These divisions require using a partial round-robin tournament to complete the season. When divisions require more than a round-robin tournament, a mirrored round-robin tournament is used for the second tournament. We may use a round-robin tournament definition and its mirror several times within a division's schedule. Table 7.7 provides examples of divisions with unbalanced schedules:

Teams in the Division	Games per season	Full Round-Robin Tournament usage	Additional Tournament rounds
10	9	1 Round-Robin	None
6	12	1 Round Robin 1 Mirrored Round-Robin	2 rounds from round-robin
7	20	2 Round-Robin 1 Mirrored Round-Robin	2 rounds from Mirrored round-robin

Table 7.7 – Unbalanced Schedule in terms of tournaments.

Each round of a tournament for a division is a tile. We apply a cost to each tile based upon the percentage of venue capacity used by each game. Tiles that contain several games with home teams in highly constrained venues will have a higher cost. A highly constrained venue is a venue where the usage rate ((number of teams/2) / daily slots), over the season, is higher than other venues. For example a venue sponsoring 6 teams, which can host 4 games a week, has a 75% usage rate. The rate suggests that the venue will be using an average of 75% of its capacity each week. A venue usage rate over 100% indicates that a feasible schedule is not possible due to venue capacity.

Tiles with the highest costs are the first tiles to be placed in the schedule. For each week the tiles placement cost is calculated by determining the number of soft constraint violations created by the tile's assignment for a given week. The week with the smallest placement cost, is chosen for the tile's placement. Tiling continues until the remaining tiles cannot be placed without causing a hard constraint violation of venue capacity.

The venue capacity constraint in relaxed to allow "over booking" of a venue. An artificial soft constraint violation weight of 9999 is used to enable the tile placement to continue.

This temporary soft constraint violation weight will highlight these games during the remaining tile placements and our local search phase.

7.6.2 Local Search Phase

The second phase involves a local search and seeks to remove hard constraint violations and reduce the number of soft constraint violations in the existing schedule. The swaps in the YSL scheduling problem are similar to those discussed by Anagnostopoulos et al. [6] for solving the TTP. All swaps are done between teams in the same division. Each swap maintains a feasible schedule for each division. The swaps are:

Home / Away Swap – We swap two games, involving the same two teams, where each team is home in one game and away in the other. The swap is done between two weeks. The impact of the swap is a reduction in the number of games for one venue and an increase for the other venue. This swap is only effective in the YSL scheduling problem for teams playing their home games at different venues.

Round Swap – All games for two rounds of a division schedule are moved between two weeks. All venues hosting a game in either week are affected by the swap.

Partial Round Swap - The partial round swap moves a set of connected games between two weeks. We begin this operation by selecting two games in different weeks. The teams in these games create our swap set. We then add teams that are playing a team in the team-swap set to that set. The process continues until all teams, in both weeks, are in the swap set or its complement. We then move all games involving teams in the swap set between weeks. We can also choose to move all teams in the complement of the swap set between weeks.

We perform the local search in a structured fashion to reach the local minima. In the first step, we consider all games at the venue during the week that has a hard constraint violation, indicated by the artificially high soft constraint violations. We calculate the reduction in soft constraint violations, if any, of performing a Home/Away swap for each game. We also calculate the reduction of software constraint violations resulting from a round swap for each round. The most beneficial swap is chosen and executed.

In the second step, we analyze all games involved in causing soft constraint violations. Each possible home and away swap, and each possible round swap for each game is analyzed for its potential reduction in soft constraint violations. All partial round swaps are identified for every round. The partial round swap considers both the swap set described above, and its complement set. The most beneficial swap among the three types of swaps is chosen for execution. The swap is performed and the resulting Master Schedule will have a lower soft constraint violation total. If no improvement is found, processing is halted as the schedule has reached its local minimum. Figure 7.1 below presents the algorithm as pseudo-code.

1. *Tile creation*
    a. For each division generate a round-robin tournament and its mirrored image.
    b. Create a tile for each round of the table until G tiles are generated, alternating the tournament table with its mirror.
    c. Assign a cost to each tile based upon the venue capacity usage of each game.
2. *Tile Placement*
    a. For each tile sorted by cost, find all weeks where the tile may be scheduled with no hard constraint violations. If no week exists, relax the venue capacity constraint and assign artificially high soft constraint violation value for that week.

b. Place all games in the tile within the week causing the fewest soft constraint violations.

3. *Local Search: Home / Away swap*
    a. Set S′ to be the schedule of swapping home and away assignments for two games with the same opponents.
    b. Consider all games with the same opponents and find S′ with the minimum soft constraint violations.
    c. If S′ reduces soft constraint violations, then swap home and away assignments for the games and continue step 3.
4. *Local Search: Round swap*
    a. Set S′ to be the schedule of swapping all games in one week with all games in another week.
    b. For each pair of weeks, calculate S′.
    c. Find the S′ that minimizes the number of soft constraint violations.
    d. If S′ improves the schedule, swap rounds and repeat step 4.
5. *Local Search: Partial Round swap*
    a. For each pair of weeks, create sets of opponents who play each other in week 1 and / or week 2.
    b. Set S′ to the schedule of swapping each set of teams between the weeks.
    c. Find the S′ that minimizes the number of soft constraint violations.
    d. If S′ improves the schedule, swap rounds and repeat step 5.

Figure 7.1 – Tiling by Round Algorithm

7.7 Results

We test our algorithm by applying it to an instance of the 2009 season of the Westchester-Putnam County Catholic Youth Organization (WPCYO) basketball program. This program has approximately 270 teams over 40 divisions. A group of 30 organizations sponsor teams across the divisions.

The hard constraints defined within the YSL scheduling problem were included in WPCYO. However, the real WPCYO had additional constraints mainly centered on team and site availability that are not specifically in the YSL scheduling problem. We do not include these items as they are easily handled by most local search approaches. However,

for sake of real world comparison, we did include these constraints in our problem instance.

Overall, we were able to improve on the actual WPCYO schedule by reducing team related soft constraint violations by 28%, and venue related soft constraint violations by 16%. The manual approach was similar in that rounds of games for a division are scheduled simultaneously. However, our selection of which tiles (or rounds) to schedule first proved to be the difference. From a venue perspective, we were not able to show much improvement due to the tight capacity constraints of the problem instance. We were only able to lower the referee category soft constraint violation by 16%. In this instance, several organizations had only one team in a referee category, rendering every home game a soft constraint violation. Table 7.8 and Table 7.9 detail the distribution of soft constraint violations for the WPCYO schedule:

Soft constraint violation Range	**WPCYO Number of Teams**	**Tiling Number of Teams**	**WPCYO Constraint Violations**	**Tiling Soft constraint violations**
40- 50	26	8	1136	352
31-40	37	14	1281	435
21-30	119	89	3034	2124
11-20	80	132	1334	1870
< 10	8	27	53	193
TOTAL	270	270	6838	4974

Table 7.8 – Venue related soft constraint violation results of WPCYO.

Soft constraint violation Range	WPCYO Number of Venues	Tiling Number of Venues	WPCYO Constraint Violations	Tiling Soft constraint violations
40- 50	2	0	86	0
32-40	8	5	282	180
22-30	2	4	48	92
12-20	12	14	186	222
< 10	6	7	42	52
TOTAL	30	30	644	546

Table 7.9 – Venue related soft constraint violation results of WPCYO.

For both tables, the column heading that include WPCYO represent the count associated with the actual WPCYO schedule. The column headings that include Tiling, represent our algorithm's output. In Table D, there were 26 teams that incurred from 40 to 50 soft constraint violations in the WPCYO schedule, while only 8 teams in our approach violated that number range of soft constraints. Conversely, only 88 teams in the WPCYO schedule had 20 or fewer team related constraint violations. The tiling approach had 159 teams with those few violations. These results would lead to more teams, coaches and players, being satisfied with the schedule.

Likewise, there were two venues with 40-50 venue related soft constraint violations in the actual schedule. There were no venues with that number of software constraint violations, using the tiling approach. This result leads to more effective and cost efficient referee assignment using our algorithm.

7.8 Conclusions

In this chapter, we propose a tiling technique that can be utilized to solve an actual Youth Sports Scheduling problem with an instance much larger than professional sports leagues.

Our approach is accomplished with the level of resources that would typically be available to a scheduler within a youth sports league.

We have also emphasized how the Youth Sports League problem occurs more often in the real world. Our problem definition accounts for standard constraints across a wide variety of youth sports leagues. This approach should enable other researchers and their approaches to be directly applied to actual instance of youth sports leagues worldwide.

CHAPTER 8 – CONCLUSIONS

8.1. SUMMARY

This thesis has introduced a tiling approach to address timetabling problems, specifically in the areas of course scheduling and sports scheduling. The tiling approach initially creates relationships between events in the problem, which are constructed through commonly know algorithms or by real world experience. These relationships are embedded in the tile and are subsequently used to quickly generate an efficient initial solution. We then use a second phase of local search to satisfy any unsatisfied constraints and tune the solution.

Our results compare favorably with the best known solutions, yet take only a fraction of their computation resources. These best known approaches use resource intensive local search techniques to perform swapping, which ultimately results in building the high cost tiles in our solution. We show how starting with tiles consisting of known event relationships can reduce these computational resources.

The ability to quickly produce a high quality solution is critical since the scheduling process itself is an iterative process. Users of the schedule look at initial drafts and modify constraints and objectives accordingly. For example, Trick in [113] describes how the production of the MLB schedule involves many rounds of schedule production and analysis. Also, new constraints about site availability may arise during the scheduling production itself. Finally, not all organizations are professional sports leagues with tremendous resources to pay for the expert consultation necessary to implement the best

known algorithms cited in this thesis. Our frugal use of resources enables organizations on a tight timeline and budget.

Beyond simply speed and performance, we show how our tiles can embed business knowledge relating to the problem. For example, Trick [113] mentions how the real MLB scheduling process involves a specialist with logistical knowledge about the traveling between the various cities. Unlike any other known scheduling approach, the tiling approach can take advantage of the scheduling practitioner's knowledge and experience. The tiles represent relationships, or conjectures, that are referenced in the solution generation without becoming a constraint.

We introduce new instances, of the sports scheduling problem, that match the problems found in the real world. We have created and solved a true TTP problem with the same characteristics as the actual MLB schedule. This type of schedule has properties that have prohibited the best known solutions from achieving a feasible solution. We have also emphasized how these scheduling properties, also included in the Youth Sports League problem, often occur in the real world. Our problem definition accounts for standard constraints across a wide variety of youth sports leagues. This approach will enable other researchers to develop better approaches that can be applied directly to solve actual instances of the youth sports leagues worldwide.

8.2. FUTURE WORK

Research into the tiling approach will continue to explore how tiling can be applied to other timetabling problems. A new instance of the TTP problem recently developed by Trick [110] [114] is one example. In this problem, the schedule instances involve bye weeks, when a team is not required to play a game. This problem would add a further dimension in tile placement by allowing the tile to be placed in a longer schedule segment than the number of games within the tile. A tile of three games could be played over four or five weeks. Also Trick [111] has introduced an umpire scheduling problem with similar constraints to the TTP.

The tile creation algorithm, using MST, may be improved to create tiles that will more likely be in the best solution. The current random method of handling ties in decision-making steps working with the team tree, could be expanded to better ascertain the proper branch of the tree to use. A local search phase could also be added to the tile creation step to further improve the set of tiles for a team. The set of tiles for a team can be viewed as a partial solution to the TTP.

We also have described a sports scheduling problem widely faced by thousands of organizations. This problem has not been reported before in the scientific literature. A set of professional-sports scheduling problems, including the TTP, have similarities to the Youth Sports League scheduling problem. However, key differences arise which include: sharing venues among several teams, significantly larger problem instances, and unbalanced schedules. These add real-world complexities that challenge current approaches. We look to document problem instances and results for the community

similar to the TTP. We are currently working with several youth sports leagues to help in defining these real world problem instances.

Finally, a goal of this research is to embed tiling functions within primitive capabilities of CSP packages, such as ILOG. These capabilities would allow problem solvers to define tiles, and associated constraint relaxations for the automatic execution of tile placement and local search phases. Incorporation of tiling into these standard problem descriptions would enhance the tiling approach within the tiling community

APPENDIX

Base Team of tile	Opp1	Opp2	Opp3	Tile Cost	Di Gaspero and Schaerf	Van Hentenryck and Vergados	Araujo et al. (GRASP)	Average
16	7	4	1	9871	3	3	2	2.67
14	7	4	1	9372	3	3	2	2.67
16	8	6	3	8635	3	3	2	2.67
16	11	10	2	8398	2	2	2	2.00
1	13	14	16	8298	3	3	3	3.00
14	8	3	6	8154	3	3	2	2.67
4	13	16	14	7758	3	3	3	3.00
2	13	16	14	7709	3	3	3	3.00
14	11	10	2	7338	2	2	2	2.00
7	13	16	14	7272	3	3	3	3.00
16	15	5	9	6861	2	2	2	2.00
10	13	16	14	6849	2	3	3	2.67
3	13	14	16	6723	3	3	3	3.00
6	13	16	14	6518	3	3	3	3.00
14	5	9	15	6256	2	2	2	2.00
13	7	4	1	6115	2	3	2	2.33
8	13	14	16	6112	3	3	3	3.00
5	13	16	14	5579	3	3	3	3.00
12	1	4	7	5379	2	3	3	2.67
9	13	16	14	5244	3	3	3	3.00
11	13	16	14	5240	3	3	3	3.00
13	8	6	3	4876	2	2	2	2.00
13	11	10	2	4793	2	2	2	2.00
12	6	3	8	4517	2	3	2	2.33

Base Team of tile	Opp1	Opp2	Opp3	Tile Cost	Di Gaspero and Schaerf	Van Hentenryck and Vergados	Araujo et al. (GRASP)	Average
2	6	3	8	4152	2	3	2	2.33
2	1	4	7	4073	2	3	3	2.67
12	13	16	14	4007	3	3	3	3.00
15	7	4	1	3995	3	3	2	2.67
1	9	15	12	3782	2	2	2	2.00
15	13	16	14	3491	3	3	1	2.33
1	11	10	2	3426	2	2	2	2.00
3	10	2	12	3305	2	2	2	2.00
2	11	15	12	3138	2	2	2	2.00
4	9	15	12	3133	2	1	1	1.33
13	15	5	9	3067	1	0	0	0.33
10	1	4	7	3024	2	2	3	2.33
10	8	3	6	2902	2	2	3	2.33
12	5	9	15	2804	2	3	2	2.33
15	11	10	2	2792	2	2	3	2.33
15	8	3	6	2786	1	2	3	2.00
8	10	2	12	2785	3	3	2	2.67
4	11	10	2	2718	2	2	2	2.00
11	1	4	7	2687	2	2	2	2.00
6	10	2	12	2616	2	3	2	2.33
2	10	5	9	2570	2	2	2	2.00
7	10	2	12	2565	2	2	3	2.33
16	13	12	14	2485	2	2	3	2.33
12	11	10	2	2474	2	3	3	2.67
9	12	2	10	2338	3	2	3	2.67
9	7	4	1	2237	2	2	2	2.00

Base Team of tile	Opp1	Opp2	Opp3	Tile Cost	Di Gaspero and Schaerf	Van Hentenryck and Vergados	Araujo et al. (GRASP)	Average
10	11	9	5	2209	3	2	2	2.33
7	11	15	9	2093	2	2	2	2.00
1	6	5	8	2042	-1	2	1	0.67
5	7	4	1	1944	-1	2	2	1.00
3	11	15	9	1859	2	-1	2	1.00
11	6	3	8	1820	3	2	1	2.00
13	16	14	12	1771	2	2	2	2.00
5	11	2	10	1682	3	2	2	2.33
14	12	13	16	1617	0	2	0	0.67
11	10	2	12	1526	3	2	1	2.00
6	11	15	9	1426	1	2	2	1.67
4	6	5	8	1396	2	2	2	2.00
8	7	4	1	1321	3	3	3	3.00
10	2	12	15	1303	1	0	1	0.67
15	9	5	12	1168	2	2	1	1.67
8	11	15	9	1158	2	3	2	2.33
9	8	3	6	1098	2	2	2	2.00
6	7	4	1	945	2	3	-2	1.00
5	9	15	12	936	1	1	1	1.00
7	6	8	5	930	2	1	2	1.67
3	7	4	1	906	1	2	-2	0.33
5	8	3	6	892	-1	-1	-2	-1.33
1	4	7	3	850	2	1	1	1.33
3	6	5	8	688	1	-1	-1	-0.33
11	5	9	15	679	-2	2	-2	-0.67
7	4	1	3	606	-2	0	0	-0.67

Base Team of tile	Opp1	Opp2	Opp3	Tile Cost	Di Gaspero and Schaerf	Van Hentenryck and Vergados	Araujo et al. (GRASP)	Average
9	5	11	15	313	-1	1	-3	-1.00
6	3	8	5	276	2	-2	-2	-0.67
4	7	3	1	270	-1	-1	1	-0.33
8	3	6	5	184	-2	1	-3	-1.33

Table A1 - Tile Comparison for 16 team instance

Base Team of tile	Opp 1	Opp2	Opp3	Tile Cost	Di Gaspero and Schaerf	Van Hentenryck and Vergados	Average
16	1	4	17	10161	2	2	2
16	19	7	20	9688	2	2	2
14	1	4	17	9685	3	2	2.5
14	19	7	20	9105	3	2	2.5
16	8	6	3	8635	2	2	2
16	11	10	2	8398	2	2	2
1	13	14	16	8298	3	2	2.5
14	8	6	3	8151	2	1	1.5
17	13	14	16	7735	3	3	3
4	13	14	16	7735	3	3	3
20	13	14	16	7550	3	2	2.5
2	13	14	16	7498	3	2	2.5
16	21	9	5	7480	1	3	2
14	11	10	2	7338	3	2	2.5
7	13	14	16	7226	3	2	2.5
19	13	14	16	7159	3	2	2.5
14	21	9	5	6921	2	2	2
3	13	14	16	6723	3	2	2.5
10	13	14	16	6680	3	2	2.5
6	13	14	16	6487	3	3	3
13	1	4	17	6412	3	2	2.5
8	13	14	16	6112	3	3	3
13	19	7	20	5903	3	2	2.5
12	1	4	17	5727	2	2	2
18	17	4	1	5451	2	2	2
16	15	18	12	5347	3	2	2.5
1	12	18	15	5240	3	2	2.5
9	13	14	16	5199	3	3	3
11	13	14	16	5137	3	3	3
12	19	7	20	4984	2	2	2
21	13	14	16	4919	3	3	3
13	8	6	3	4876	3	2	2.5
13	11	10	2	4793	3	3	3
18	19	7	20	4750	2	2	2
4	12	18	15	4552	3	3	3
17	12	18	15	4552	3	2	2.5
12	8	6	3	4433	3	3	3
2	9	21	22	4413	2	2	2

Base Team of tile	Opp 1	Opp2	Opp3	Tile Cost	Di Gaspero and Schaerf	Van Hentenryck and Vergados	Average
20	12	18	15	4316	3	3	3
15	1	4	17	4303	2	2	2
2	1	4	20	4270	2	-1	0.5
5	14	13	16	4235	3	2	2.5
14	12	18	22	4093	2	1	1.5
2	8	6	3	4045	2	3	2.5
22	13	14	16	4034	3	1	2
18	8	6	3	4033	2	2	2
22	1	4	17	3978	3	2	2.5
7	15	18	12	3922	2	1	1.5
3	15	18	12	3849	3	2	2.5
19	15	18	12	3827	3	2	2.5
12	13	14	16	3777	3	2	2.5
22	11	10	2	3765	3	3	3
2	19	7	17	3751	2	2	2
15	19	7	20	3739	2	1	1.5
2	15	18	12	3733	3	1	2
13	21	9	5	3697	3	2	2.5
22	19	7	20	3662	3	2	2.5
15	13	14	16	3441	3	1	2
1	11	10	2	3426	3	2	2.5
12	21	9	5	3365	2	3	2.5
10	1	4	17	3363	3	2	2.5
6	15	18	12	3350	3	2	2.5
18	13	14	16	3166	3	2	2.5
8	15	18	12	3125	3	2	2.5
10	9	21	22	3107	1	3	2
1	9	21	22	3100	2	3	2.5
11	1	4	17	3029	3	2	2.5
10	15	18	12	2896	2	1	1.5
18	21	9	5	2856	2	2	2
15	11	10	2	2792	3	3	3
10	8	6	3	2791	1	2	1.5
3	11	10	2	2788	2	3	2.5
15	8	6	3	2775	2	2	2
17	11	10	2	2718	3	2	2.5
4	11	10	2	2718	3	2	2.5
18	11	10	2	2702	1	1	1

Base Team of tile	Opp 1	Opp2	Opp3	Tile Cost	Di Gaspero and Schaerf	Van Hentenryck and Vergados	Average
21	20	4	1	2689	2	2	2
10	19	7	20	2651	2	1	1.5
17	9	21	22	2564	3	2	2.5
4	9	21	22	2564	3	2	2.5
9	1	4	17	2548	3	2	2.5
22	8	6	3	2537	3	3	3
21	11	10	2	2494	3	3	3
12	11	10	2	2474	1	2	1.5
20	11	10	2	2471	3	2	2.5
22	15	18	12	2468	3	-1	1
20	9	21	22	2405	3	2	2.5
16	14	13	22	2388	1	1	1
5	12	18	15	2352	3	2	2.5
21	19	7	17	2318	2	2	2
11	19	7	20	2311	1	1	1
5	1	4	17	2263	3	1	2
8	2	10	11	2259	3	2	2.5
21	15	18	12	2244	3	1	2
7	9	21	22	2142	2	1	1.5
9	15	18	12	2134	3	2	2.5
19	9	21	22	2104	2	3	2.5
6	11	10	2	2089	3	2	2.5
7	2	10	11	2082	2	-2	0
1	6	8	5	2001	2	-1	0.5
14	16	13	15	1993	1	-1	0
19	11	10	2	1971	3	2	2.5
9	19	7	20	1965	3	1	2
9	11	10	2	1845	3	3	3
12	18	15	22	1796	-1	2	0.5
13	22	18	12	1753	2	1	1.5
11	8	6	3	1730	2	1	1.5
22	21	9	5	1714	3	-2	0.5
11	15	18	12	1677	1	-2	-0.5
5	2	10	11	1671	-1	-1	-1
5	19	7	20	1644	3	2	2.5
8	1	4	17	1592	-1	2	0.5
3	9	21	22	1559	2	3	2.5
15	21	9	5	1559	3	1	2

Base Team of tile	Opp 1	Opp2	Opp3	Tile Cost	Di Gaspero and Schaerf	Van Hentenryck and Vergados	Average
13	15	14	16	1395	2	2	2
4	6	8	5	1378	1	2	1.5
6	9	21	22	1367	3	2	2.5
2	11	10	5	1305	-1	1	0
17	3	8	5	1287	2	-1	0.5
21	8	6	3	1278	2	1	1.5
6	1	4	17	1236	3	1	2
1	3	19	7	1217	2	1	1.5
11	9	21	22	1215	1	1	1
20	6	8	5	1196	2	-1	0.5
8	19	7	20	1168	2	2	2
11	5	10	2	1096	1	2	1.5
9	8	6	3	1057	3	2	2.5
3	1	4	20	1012	-2	2	0
8	9	21	22	969	3	2	2.5
3	19	7	17	963	2	-1	0.5
7	3	8	5	947	2	-1	0.5
18	12	15	22	938	-2	-1	-1.5
10	2	11	5	912	-3	-2	-2.5
15	22	18	12	886	1	-2	-0.5
5	3	6	8	826	2	1	1.5
1	4	20	17	756	1	-1	0
19	8	3	5	742	-1	2	0.5
6	19	7	20	723	2	2	2
19	1	4	20	688	1	1	1
17	7	19	6	676	2	-1	0.5
5	9	21	22	640	-1	1	0
3	6	8	5	622	2	-3	-0.5
7	1	4	20	570	2	1	1.5
4	7	19	3	549	-1	-1	-1
9	5	21	22	341	-3	-1	-2
20	7	19	3	339	-2	-1	-1.5
21	5	9	22	327	-2	-2	-2
6	3	8	5	276	-2	-1	-1.5
20	1	4	17	264	-2	-2	-2
8	3	6	5	184	-3	-2	-2.5
19	6	7	17	84	1	1	1
7	6	19	17	36	-1	-1	-1

Base Team of tile	Opp 1	Opp2	Opp3	Tile Cost	Di Gaspero and Schaerf	Van Hentenryck and Vergados	Average
4	17	20	1	0	-2	-3	-2.5
17	20	4	1	0	-3	-3	-3

Table A2 - Tile Comparison for 22 team instance

Base Team of tile	Opp1	Opp2	Opp3	Tile Cost	Van Hentenryck and Vergados	Di Gaspero and Schaerf	Average
16	20	17	4	10117	2	1	2.00
16	19	7	1	9600	2	2	1.00
14	20	4	1	9569	0	2	0.00
16	10	26	2	9338	3	3	1.00
14	19	7	17	9088	0	0	0.00
16	8	6	3	8635	2	2	1.00
1	13	14	16	8298	2	3	0.67
14	10	26	2	8162	3	3	1.00
14	8	6	3	8151	0	2	0.00
4	13	14	16	7735	3	2	1.50
17	13	14	16	7735	3	2	1.50
16	5	11	25	7615	2	0	0.00
20	13	14	16	7550	3	3	1.00
2	13	14	16	7498	3	3	1.00
16	21	9	23	7384	2	1	2.00
7	13	14	16	7226	3	3	1.00
19	13	14	16	7159	3	1	3.00
14	21	9	23	6872	0	-1	0.00
14	5	11	25	6842	0	1	0.00
26	13	14	16	6777	3	2	1.50
3	13	14	16	6723	3	2	1.50
10	13	14	16	6680	3	3	1.00
6	13	14	16	6487	3	3	1.00
25	13	14	16	6472	3	3	1.00
13	20	4	1	6302	0	2	0.00
8	16	14	13	6112	3	3	1.00
13	19	7	17	5880	0	0	0.00
13	10	26	2	5822	3	3	1.00
16	15	22	24	5817	0	-1	0.00
23	13	14	16	5797	3	2	1.50

Base Team of tile	Opp1	Opp2	Opp3	Tile Cost	Van Hentenryck and Vergados	Di Gaspero and Schaerf	Average
14	22	24	15	5657	2	2	1.00
12	20	4	1	5596	2	2	1.00
5	13	14	16	5526	3	3	1.00
18	20	17	4	5337	2	2	1.00
1	15	18	12	5240	2	2	1.00
9	13	14	16	5199	3	3	1.00
11	13	14	16	5137	3	2	1.50
2	21	24	22	5020	2	2	1.00
12	19	7	17	4982	2	3	0.67
22	10	26	2	4969	3	2	1.50
21	13	14	16	4919	3	3	1.00
13	8	6	3	4876	0	2	0.00
18	19	7	1	4732	1	-3	-0.33
4	15	18	12	4552	2	2	1.00
17	15	18	12	4552	3	2	1.50
24	10	26	2	4420	3	2	1.50
20	15	18	12	4316	2	2	1.00
1	10	26	2	4272	2	3	0.67
2	20	4	1	4270	2	2	1.00
26	21	24	22	4228	2	2	1.00
15	20	4	1	4188	3	2	1.50
2	8	6	3	4045	2	2	1.00
22	13	14	16	4034	3	3	1.00
18	8	6	3	4033	0	-2	0.00
12	9	6	3	3976	2	0	0.00
12	5	8	23	3945	3	2	1.50
7	15	18	12	3922	3	2	1.50
15	10	26	2	3916	3	3	1.00
22	4	1	17	3892	3	2	1.50
2	5	9	23	3872	0	0	0.00
3	15	18	12	3849	3	3	1.00
13	5	11	25	3849	0	1	0.00
26	20	17	1	3828	3	2	1.50
19	15	18	12	3827	1	1	1.00
3	10	26	2	3825	3	3	1.00
12	13	14	16	3777	3	1	3.00
2	19	7	17	3751	0	2	0.00
2	15	18	12	3733	2	2	1.00
15	19	7	17	3721	2	0	0.00

Base Team of tile	Opp1	Opp2	Opp3	Tile Cost	Van Hentenryck and Vergados	Di Gaspero and Schaerf	Average
10	21	24	22	3712	0	3	0.00
21	10	26	2	3695	3	3	1.00
22	19	7	20	3662	2	2	1.00
13	21	9	23	3614	0	-1	0.00
23	10	26	2	3581	3	2	1.50
4	10	26	2	3570	3	3	1.00
17	10	26	2	3570	3	2	1.50
12	24	21	22	3543	2	2	1.00
18	10	26	2	3502	3	2	1.50
15	13	14	16	3441	3	2	1.50
26	8	6	3	3400	2	2	1.00
6	15	18	12	3350	3	2	1.50
8	10	26	2	3349	3	3	1.00
20	10	26	2	3323	3	3	1.00
1	21	24	22	3257	2	2	1.00
26	19	7	4	3253	3	2	1.50
10	1	17	20	3233	3	2	1.50
18	13	14	16	3166	3	2	1.50
8	15	18	12	3125	2	2	1.00
26	5	9	23	3112	2	-1	-2.00
6	10	26	2	3105	3	2	1.50
18	22	24	21	3057	2	3	0.67
18	9	5	23	3039	2	2	1.00
9	10	26	2	3033	3	3	1.00
26	15	18	12	2996	0	3	0.00
24	1	4	17	2965	3	0	0.00
23	15	18	12	2964	3	2	1.50
7	10	26	2	2943	3	3	1.00
10	15	18	12	2896	3	2	1.50
25	15	18	12	2872	3	1	3.00
19	10	26	2	2829	3	2	1.50
5	10	26	2	2818	3	1	3.00
4	21	24	22	2814	2	3	0.67
17	21	24	22	2814	2	3	0.67
10	8	6	3	2791	2	2	1.00
15	8	6	3	2775	3	2	1.50
1	5	11	25	2769	2	2	1.00
24	15	18	12	2749	3	0	#DIV/0!
20	21	24	22	2694	3	2	1.50

Base Team of tile	Opp1	Opp2	Opp3	Tile Cost	Van Hentenryck and Vergados	Di Gaspero and Schaerf	Average
21	20	17	1	2689	2	3	0.67
24	19	7	20	2685	2	2	1.00
10	19	7	4	2648	3	2	1.50
25	21	24	22	2592	2	-1	-2.00
11	7	20	17	2571	2	3	0.67
10	5	9	23	2561	2	0	0.00
9	17	4	1	2548	3	2	1.50
7	21	24	22	2498	3	2	1.50
19	21	24	22	2483	0	2	0.00
22	15	18	12	2468	3	1	3.00
22	23	6	3	2342	0	0	0.00
21	19	7	4	2318	2	2	1.00
1	8	23	9	2292	-2	0	0.00
21	15	18	12	2244	3	3	1.00
11	10	26	2	2193	3	3	1.00
13	15	22	24	2190	2	1	2.00
5	20	4	1	2143	3	2	1.50
22	5	8	25	2135	0	3	0.00
9	15	18	12	2134	1	3	0.33
4	5	11	25	2049	2	1	2.00
17	5	11	25	2049	2	2	1.00
11	4	1	22	2028	0	2	0.00
24	13	14	6	1967	3	2	1.50
9	19	7	20	1965	2	2	1.00
23	4	17	1	1927	2	3	0.67
11	6	3	19	1819	2	-2	-1.00
20	5	11	25	1804	0	0	0.00
25	19	17	1	1750	2	2	1.00
24	5	11	25	1743	-1	2	-0.50
4	8	23	9	1722	2	2	1.00
17	8	23	9	1722	2	3	0.67
6	21	24	22	1707	2	2	1.00
3	21	24	22	1694	2	2	1.00
15	5	11	25	1685	0	2	0.00
11	15	18	12	1677	3	2	1.50
25	20	4	7	1677	3	0	0.00
3	5	11	25	1632	2	-1	-2.00
5	19	7	17	1631	2	-1	-2.00
5	22	15	18	1598	2	1	2.00

Base Team of tile	Opp1	Opp2	Opp3	Tile Cost	Van Hentenryck and Vergados	Di Gaspero and Schaerf	Average
20	8	23	9	1561	2	-1	-2.00
24	8	6	3	1559	-2	2	-1.00
23	19	7	20	1552	1	3	0.33
8	20	4	17	1530	3	2	1.50
15	21	9	23	1510	-1	1	-1.00
25	8	6	3	1436	1	2	0.50
25	5	9	23	1396	2	2	1.00
7	8	5	9	1380	1	2	0.50
1	19	6	3	1372	2	-1	-2.00
21	8	6	3	1278	3	2	1.50
11	9	21	24	1269	-1	0	0.00
8	21	24	22	1257	2	2	1.00
19	5	9	25	1217	2	2	1.00
7	23	11	25	1171	0	0	0.00
19	8	23	11	1139	3	2	1.50
6	20	4	1	1122	2	2	1.00
21	5	11	25	1041	1	-2	-0.50
8	9	11	25	1014	-1	2	-0.50
3	20	4	1	1012	2	0	0.00
6	5	11	25	995	2	-1	-2.00
8	7	19	1	984	3	0	0.00
9	21	24	22	933	2	-3	-0.67
23	21	24	22	883	3	1	3.00
5	8	6	3	826	-1	0	0.00
3	8	9	23	820	2	1	2.00
23	5	9	11	785	1	-2	-0.50
4	19	6	3	751	-1	2	-0.50
17	19	6	3	751	0	-3	0.00
19	20	4	17	688	2	2	1.00
5	9	21	24	622	1	-1	-1.00
20	19	6	3	578	1	-1	-1.00
6	8	23	9	547	-1	-1	1.00
12	15	18	12	456	-1	-1	1.00
9	5	11	25	431	-1	2	-0.50
7	19	6	3	339	-1	2	-0.50

Table A3 - Tile Comparison for 26 team instance

TEAM	PIT	CHC	CIN	STL	HOU	MIL
PIT	0	409	259	558	1152	446
CHC	409	0	240	258	954	82
CIN	259	240	0	304	910	313
STL	558	258	304	0	696	323
HOU	1152	954	910	696	0	1017
MIL	446	82	313	323	1017	0

Table A4 - Table Distance Matrix for DIV 1 Instance

H/A	PIT	CHC	CIN	STL	HOU	MIL
PIT	0	3	2	3	2	3
CHC	3	0	3	2	3	2
CIN	2	2	0	3	3	3
STL	2	2	3	0	3	3
HOU	3	3	2	3	0	2
MIL	3	3	3	2	2	0

Table A5 - Match-Ups for DIV 1 Instance

TEAM	PIT	CHC	CIN	STL	HOU	MIL
Tile 1	CIN,CHC,MIL	PIT,CIN,MIL	STL,HOU,MIL	MIL,CIN,PIT	MIL,CIN,PIT	PIT,CHC,CIN
Tile 2	CIN,CHC,MIL	PIT,CIN,MIL	STL,HOU,MIL	CHC,HOU	CHC,STL	PIT,CHC,CIN
Tile 3	STL,HOU	STL,HOU	PIT,CHC	CHC,HOU	CHC,STL	STL,HOU
Tile 4	STL,HOU	STL,HOU	PIT,CHC	PIT,CIN,MIL	PIT,CIN,MIL	STL,HOU
Tile 5	CHC,HOU,MIL	PIT,HOU,MIL	CHC,STL,MIL	PIT,CIN,HOU	PIT,CIN,STL	PIT,CIN,STL

Table A6 - Tiles for DIV 1 Instance

TEAM	W1	W2	W3	W4	W5	W6	W7	W8	W9	W10	W11	W12	W13
PIT	CHC	MIL	-HOU	-STL	-MIL	CHC	CIN	-CHC	-STL	HOU	CHC	-MIL	-CHC
CHC	-PIT	-STL	CIN	HOU	STL	-PIT	MIL	PIT	-HOU	-CIN	-PIT	HOU	PIT
CIN	MIL	-HOU	-CHC	-MIL	HOU	STL	-PIT	-STL	-MIL	CHC	MIL	STL	-MIL
STL	-HOU	CHC	MIL	PIT	-CHC	-CIN	-HOU	CIN	PIT	MIL	-HOU	-CIN	HOU
HOU	STL	CIN	PIT	-CHC	-CIN	-MIL	STL	MIL	CHC	-PIT	STL	-CHC	-STL
MIL	-CIN	-PIT	-STL	CIN	PIT	HOU	-CHC	-HOU	CIN	-STL	-CIN	PIT	CIN

Table A7 - Solution for DIV 1 Instance (- indicates home game) – Part I

TEAM	W14	W15	W16	W17	W18	W19	W20	W21	W22	W23	W24	W25	W26
PIT	HOU	STL	MIL	-CHC	-STL	-CIN	HOU	STL	MIL	-HOU	-CIN	-MIL	CIN
CHC	-MIL	-CIN	-HOU	PIT	CIN	MIL	-CIN	-HOU	-STL	MIL	HOU	STL	-MIL
CIN	-STL	CHC	STL	-HOU	-CHC	PIT	CHC	MIL	-HOU	-STL	PIT	HOU	-PIT
STL	CIN	-PIT	-CIN	-MIL	PIT	HOU	-MIL	-PIT	CHC	CIN	-MIL	-CHC	HOU
HOU	-PIT	MIL	CHC	CIN	-MIL	-STL	-PIT	CHC	CIN	PIT	-CHC	-CIN	-STL
MIL	CHC	-HOU	-PIT	STL	HOU	-CHC	STL	-CIN	-PIT	-CHC	STL	PIT	CHC

Table A8 - Solution for DIV 1 Instance (- indicates home game) – Part II

TEAM	WAS	NYM	PHI	ATL	FLA	ARZ	LAD	SF	SD	COL
WAS	0	205	139	527	933	1981	2308	2440	2275	1480
NYM	205	0	66	732	1104	2141	2457	2569	2431	1616
PHI	139	66	0	666	1047	2090	2409	2528	2381	1572
ATL	527	732	666	0	617	1598	1950	2140	1895	1200
FLA	933	1104	1047	617	0	1974	2339	2583	2258	1710
ARZ	1981	2141	2090	1598	1974	0	365	645	298	593
LAD	2308	2457	2409	1950	2339	365	0	334	124	851
SF	2440	2569	2528	2140	2583	645	334	0	458	961
SD	2275	2431	2381	1895	2258	298	124	458	0	847
COL	1480	1616	1572	1200	1710	593	851	961	847	0

Table A9 - Table Distance Matrix for DIV 2 Instance

H/A	WAS	NYM	PHI	ATL	FLA	ARZ	LAD	SF	SD	COL
WAS	0	2	2	2	2	1	1	1	1	1
NYM	2	0	2	2	2	1	1	1	1	1
PHI	2	2	0	2	2	1	1	1	1	1
ATL	2	2	2	0	2	1	1	1	1	1
FLA	2	2	2	2	0	1	1	1	1	1
ARZ	1	1	1	1	1	0	2	2	2	2
LAD	1	1	1	1	1	2	0	2	2	2
SF	1	1	1	1	1	2	2	0	2	2
SD	1	1	1	1	1	2	2	2	0	2
COL	1	1	1	1	1	2	2	2	2	0

Table A10 - Match-Ups for DIV 2 Instance

TEAM	WAS	NYM	PHI	ATL	FLA
Tile1	NYM,PHI,FLA	SD,LAD,SF	SD,LAD,SF	SD,LAD,SF	SD,LAD,SF
Tile2	SD,LAD,SF	ARZ,COL,ATL	ATL,COL,ARZ	FLA,COL,ARZ	ATL,COL,ARZ
Tile3	ARZ,COL,ATL	PHI,WAS,FLA	WAS,NYM,FLA	WAS,PHI,NYM	WAS,NYM,PHI
Tile4	NYM,PHI	WAS,PHI	WAS,NYM	PHI,NYM	NYM,PHI
Tile5	ATL,FLA	ATL,FLA	ATL,FLA	WAS,FLA	WAS,ATL

Table A11 - Tiles for DIV 2 Instance Part I

TEAM	ARZ	LAD	SF	SD	COL
Tile 1	WAS,PHI,NYM	WAS,PHI,NYM	WAS,PHI,NYM	WAS,PHI,NYM	SD,LAD,SF
Tile 2	COL,ATL,FLA	COL,ATL,FLA	COL,ATL,FLA	COL,ATL,FLA	ARZ,ATL,FLA
Tile 3	SD,LAD,SF	ARZ,SD,SF	LAD,SD,ARZ	SF,LAD,ARZ	WAS,PHI,NYM
Tile 4	LAD,SF	ARZ,COL	ARZ,COL	SF,LAD	LAD,SF
Tile 5	SD,COL	SF,SD	LAD,SD	ARZ,COL	ARZ,SD

Table A12 - Tiles for DIV 2 Instance Part II

TEAM	W1	W2	W3	W4	W5	W6	W7	W8	W9	W10	W11	W12	W13
WAS	ATL	FLA	-NYM	ARZ	ATL	-COL	-PHI	-ARZ	SF	LAD	SD	-FLA	NYM
NYM	FLA	ATL	WAS	-SD	-COL	SD	SF	LAD	-ATL	-PHI	ATL	-ARZ	-WAS
PHI	ARZ	COL	-SD	-ATL	FLA	ATL	WAS	-ATL	-FLA	NYM	-ARZ	SF	LAD
ATL	-WAS	-NYM	-FLA	PHI	-WAS	-PHI	-ARZ	PHI	NYM	FLA	-NYM	-LAD	FLA
FLA	-NYM	-WAS	ATL	-COL	-PHI	-ARZ	SD	SF	PHI	-ATL	-LAD	WAS	-ATL
ARZ	-PHI	SF	LAD	-WAS	-LAD	FLA	ATL	WAS	-COL	-SF	PHI	NYM	COL
LAD	COL	SD	-ARZ	SF	ARZ	-SF	COL	-NYM	-SD	-WAS	FLA	ATL	-PHI
SF	-SD	-ARZ	COL	-LAD	SD	LAD	-NYM	-FLA	-WAS	ARZ	COL	-PHI	-SD
SD	SF	-LAD	PHI	NYM	-SF	-NYM	-FLA	COL	LAD	-COL	-WAS	COL	SF
COL	-LAD	-PHI	-SF	FLA	NYM	WAS	-LAD	-SD	ARZ	SD	-SF	-SD	-ARZ

Table A13 - Solution for DIV 2 Instance (- indicates home game) – Part I

TEAM	W14	W15	W16	W17	W18	W19	W20	W21	W22	W23	W24	W25	W26
WAS	-FLA	-NYM	-SD	PHI	FLA	COL	-LAD	NYM	-ATL	-PHI	-SF	PHI	-ATL
NYM	-ATL	WAS	-FLA	ARZ	COL	-LAD	-PHI	-WAS	PHI	-FLA	PHI	-SF	FLA
PHI	SD	-FLA	ATL	-WAS	-LAD	FLA	NYM	-COL	-NYM	WAS	-NYM	-WAS	-SF
ATL	NYM	-SD	-PHI	LAD	SD	SF	-COL	-SF	WAS	COL	ARZ	-FLA	WAS
FLA	WAS	PHI	NYM	-SD	-WAS	-PHI	ARZ	LAD	-SF	NYM	COL	ATL	-NYM
ARZ	-LAD	-SF	COL	-NYM	SF	SD	-FLA	-SD	LAD	SD	-ATL	-SD	-COL
LAD	ARZ	-COL	-SF	-ATL	PHI	NYM	WAS	-FLA	-ARZ	SF	SD	-COL	-SD
SF	-COL	ARZ	LAD	-COL	-ARZ	-ATL	SD	ATL	FLA	-LAD	WAS	NYM	PHI
SD	-PHI	ATL	WAS	FLA	-ATL	-ARZ	-SF	ARZ	-COL	-ARZ	-LAD	ARZ	LAD
COL	SF	LAD	-ARZ	SF	-NYM	-WAS	ATL	PHI	SD	-ATL	-FLA	LAD	ARZ

Table A14 - Solution for DIV 2 Instance (- indicates home game) – Part II

TEAM	WAS	NYM	PHI	ATL	FLA	ARZ	LAD	SF	SD
WAS	0	205	139	527	933	1981	2308	2440	2275
NYM	205	0	66	732	1104	2141	2457	2569	2431
PHI	139	66	0	666	1047	2090	2409	2528	2381
ATL	527	732	666	0	617	1598	1950	2140	1895
FLA	933	1104	1047	617	0	1974	2339	2583	2258
ARZ	1981	2141	2090	1598	1974	0	365	645	298
LAD	2308	2457	2409	1950	2339	365	0	334	124
SF	2440	2569	2528	2140	2583	645	334	0	458
SD	2275	2431	2381	1895	2258	298	124	458	0
COL	1480	1616	1572	1200	1710	593	851	961	847
BOS	394	189	255	921	1270	2295	2600	2695	2581
NYY	205	0	66	732	1104	2141	2457	2569	2431
BAL	30	176	109	557	960	2001	2325	2454	2294
TB	803	992	929	418	207	1801	2165	2398	2090
TOR	358	345	338	724	1249	1888	2181	2267	2168
CLE	310	409	369	536	1094	1741	2050	2160	2028
DET	408	496	461	575	1157	1670	1973	2074	1954
CWS	598	712	672	579	1195	1452	1752	1857	1734
MIN	935	1016	988	900	1515	1278	1531	1586	1535
KC	936	1096	1046	676	1241	1047	1364	1504	1335

Table A15 - Table Distance Matrix for DIV 4 Instance – PART I

TEAM	COL	BOS	NYY	BAL	TB	TOR	CLE	DET	CWS	MIN	KC
WAS	1480	394	205	30	803	358	310	408	598	935	936
NYM	1616	189	0	176	992	345	409	496	712	1016	1096
PHI	1572	255	66	109	929	338	369	461	672	988	1046
ATL	1200	921	732	557	418	724	536	575	579	900	676
FLA	1710	1270	1104	960	207	1249	1094	1157	1195	1515	1241
ARZ	593	2295	2141	2001	1801	1888	1741	1670	1452	1278	1047
LAD	851	2600	2457	2325	2165	2181	2050	1973	1752	1531	1364
SF	961	2695	2569	2454	2398	2267	2160	2074	1857	1586	1504
SD	847	2581	2431	2294	2090	2168	2028	1954	1734	1535	1335
COL	0	1753	1616	1495	1510	1331	1207	1126	906	689	544
BOS	1753	0	189	364	1170	430	554	628	848	1119	1248
NYY	1616	189	0	176	992	345	409	496	712	1016	1096
BAL	1495	364	176	0	832	344	314	411	607	939	962
TB	1510	1170	992	832	0	1091	922	977	997	1311	1034
TOR	1331	430	345	344	1091	0	199	226	437	689	849
CLE	1207	554	409	314	922	199	0	98	303	626	694
DET	1126	628	496	411	977	226	98	0	220	529	627
CWS	906	848	712	607	997	437	303	220	0	353	414
MIN	689	1119	1016	939	1311	689	626	529	353	0	413
KC	544	1248	1096	962	1034	849	694	627	414	413	0

Table A16 - Table Distance Matrix for DIV 4 Instance – PART II

H/A	WAS	NYM	PHI	ATL	FLA	ARZ	LAD	SF	SD	COL
WAS	0	2	2	2	2	1	1	1	1	1
NYM	2	0	2	2	2	1	1	1	1	1
PHI	2	2	0	2	2	1	1	1	1	1
ATL	2	2	2	0	2	1	1	1	1	1
FLA	2	2	2	2	0	1	1	1	1	1
ARZ	1	1	1	1	1	0	2	2	2	2
LAD	1	1	1	1	1	2	0	2	2	2
SF	1	1	1	1	1	2	2	0	2	2
SD	1	1	1	1	1	2	2	2	0	2
COL	1	1	1	1	1	2	2	2	2	0
BOS	1	1	1	1	1	0	0	0	0	0
NYY	1	1	1	1	1	0	0	0	0	0
BAL	1	1	1	1	1	0	0	0	0	0
TB	1	1	1	1	1	0	0	0	0	0
TOR	1	1	1	1	1	0	0	0	0	0
CLE	0	0	0	0	0	1	1	1	1	1
DET	0	0	0	0	0	1	1	1	1	1
CWS	0	0	0	0	0	1	1	1	1	1
MIN	0	0	0	0	0	1	1	1	1	1
KC	0	0	0	0	0	1	1	1	1	1

Table A17 - Match-Ups for DIV 4 Instance – Part I

H/A	BOS	NYY	BAL	TB	TOR	CLE	DET	CWS	MIN	KC
WAS	1	1	1	1	1	0	0	0	0	0
NYM	1	1	1	1	1	0	0	0	0	0
PHI	1	1	1	1	1	0	0	0	0	0
ATL	1	1	1	1	1	0	0	0	0	0
FLA	1	1	1	1	1	0	0	0	0	0
ARZ	0	0	0	0	0	1	1	1	1	1
LAD	0	0	0	0	0	1	1	1	1	1
SF	0	0	0	0	0	1	1	1	1	1
SD	0	0	0	0	0	1	1	1	1	1
COL	0	0	0	0	0	1	1	1	1	1
BOS	0	2	2	2	2	1	1	1	1	1
NYY	2	0	2	2	2	1	1	1	1	1
BAL	2	2	0	2	2	1	1	1	1	1
TB	2	2	2	0	2	1	1	1	1	1
TOR	2	2	2	2	0	1	1	1	1	1
CLE	1	1	1	1	1	0	2	2	2	2
DET	1	1	1	1	1	2	0	2	2	2
CWS	1	1	1	1	1	2	2	0	2	2
MIN	1	1	1	1	1	2	2	2	0	2
KC	1	1	1	1	1	2	2	2	2	0

Table A18 - Match-Ups for DIV 4 Instance – Part I

TEAM	WAS	NYM	PHI	ATL	FLA
Tile1	SD,LAD,SF	SD,LAD,SF	SD,LAD,SF	WAS,COL,ARZ	SD,LAD,SF
Tile2	ATL,COL,ARZ	ATL,COL,ARZ	ATL,COL,ARZ	SD,LAD,SF	ATL,COL,ARZ
Tile3	ATL,TB,FLA	ATL,TB,FLA	ATL,TB,FLA	NYM,BOS,NYY	NYM,NYY,BOS
Tile4	NYM,BOS,NYY	PHI,BAL,WAS	NYM,BAL,WAS	BAL,PHI,TOR	BAL,PHI,TOR
Tile5	BAL,PHI,TOR	NYY,TOR,BOS	NYY,TOR,BOS	FLA,TB,WAS	TB,ATL,WAS
Tile6	PHI,NYM,FLA	PHI,WAS,FLA	NYM,WAS,FLA	NYM,PHI,FLA	NYM,PHI,WAS

Table A19 - Tiles for DIV 4 Instance Part I

TEAM	ARZ	LAD	SF	SD	COL
Tile1	SD,LAD,SF	SD,ARZ,SF	LAD,SD,ARZ	ARZ,LAD,SF	SD,LAD,SF
Tile2	WAS,PHI,NYM	WAS,PHI,NYM	WAS,PHI,NYM	WAS,PHI,NYM	WAS,ATL,FLA
Tile3	CLE,ATL,FLA	CLE,ATL,FLA	CLE,ATL,FLA	CLE,ATL,FLA	CLE,PHI,NYM
Tile4	MIN,CWS,DET	MIN,CWS,DET	MIN,CWS,DET	MIN,CWS,DET	MIN,CWS,DET
Tile5	SD,COL,KC	SD,COL,KC	SD,COL,KC	ARZ,COL,KC	KC,ARZ,SD
Tile6	SF,LAD,COL	SF,ARZ,COL	LAD,ARZ,COL	LAD,SF,COL	SF,LAD,ARZ

Table A20 - Tiles for DIV 4 Instance Part II

TEAM	BOS	NYY	BAL	TB	TOR
Tile1	NYM,NYY,BAL	NYM,BOS,BAL	NYM,BOS,NYY	TOR,BOS,NYY	ATL,TB,FLA
Tile2	PHI,BAL,WAS	PHI,BAL,WAS	PHI,NYY,WAS	ATL,FLA,BAL	NYM,NYY,BOS
Tile3	ATL,TB,FLA	ATL,TB,FLA	ATL,TB,FLA	WAS,BAL,PHI	CWS,MIN,KC
Tile4	TOR,CLE,DET	TOR,CLE,DET	TOR,CLE,DET	NYM,BOS,NYY	WAS,BAL,PHI
Tile5	CWS,MIN,KC	CWS,MIN,KC	CWS,MIN,KC	TOR,CLE,DET	BOS,NYY,BAL
Tile6	NYY,TB,TOR	BOS,TB,TOR	BOS,TB,TOR	CWS,MIN,KC	CLE,DET,TB

Table A21 - Tiles for DIV 4 Instance Part III

TEAM	CLE	DET	CWS	MIN	KC
Tile1	CWS,MIN,KC	CWS,MIN,KC	DET,MIN,KC	DET,CWS,KC	DET,CWS,MIN
Tile2	DET,TB,TOR	CLE,TB,TOR	CLE,TB,TOR	CLE,TB,TOR	CLE,TB,TOR
Tile3	BOS,NYY,BAL	BOS,NYY,BAL	BOS,NYY,BAL	BOS,NYY,BAL	BOS,NYY,BAL
Tile4	DET,CWS,MIN	CLE,CWS,MIN	CLE,DET,MIN	CLE,DET,CWS	CLE,DET,CWS
Tile5	KC,COL,ARZ	KC,COL,ARZ	KC,COL,ARZ	KC,COL,ARZ	MIN,COL,ARZ
Tile6	SD,LAD,SF	SD,LAD,SF	SD,LAD,SF	SD,LAD,SF	SD,LAD,SF

Table A22 - Tiles for DIV 4 Instance Part IV

TEAM	W1	W2	W3	W4	W5	W6	W7	W8	W9	W10	W11	W12
WAS	SF	LAD	ARZ	-FLA	SD	COL	PHI	-ATL	FLA	ATL	-SD	TB
NYM	BOS	-BAL	-FLA	PHI	-ATL	LAD	SF	COL	-LAD	BAL	-FLA	-SD
PHI	FLA	ATL	-BOS	-NYM	BAL	-ATL	-WAS	NYY	-ATL	-SD	ARZ	LAD
ATL	BAL	-PHI	-NYY	BOS	NYM	PHI	-FLA	WAS	PHI	-WAS	-TB	FLA
FLA	-PHI	-TOR	NYM	WAS	-SF	-NYY	ATL	TB	-WAS	-LAD	NYM	-ATL
ARZ	-CWS	-KC	-WAS	LAD	-COL	-SD	-DET	SD	-COL	-SF	-PHI	SF
LAD	-COL	-WAS	KC	-ARZ	-CLE	-NYM	SD	-SF	NYM	FLA	CWS	-PHI
SF	-WAS	-CWS	-COL	KC	FLA	-CLE	-NYM	LAD	SD	ARZ	-COL	-ARZ
SD	KC	MIIN	DET	-COL	-WAS	ARZ	-LAD	-ARZ	-SF	PHI	WAS	NYM
COL	LAD	-DET	SF	SD	ARZ	-WAS	-MIIN	-NYM	ARZ	-CWS	SF	KC
BOS	-NYM	-CLE	PHI	-ATL	NYY	DET	-BAL	CLE	BAL	-TB	-MIIN	-BAL
NYY	-CLE	TB	ATL	-TB	-BOS	FLA	TB	-PHI	TOR	DET	CLE	-TOR
BAL	-ATL	NYM	TOR	CLE	-PHI	-TB	BOS	TOR	-BOS	-NYM	-TOR	BOS
TB	-TOR	-NYY	-CLE	NYY	TOR	BAL	-NYY	-FLA	DET	BOS	ATL	-WAS
TOR	TB	FLA	-BAL	-MIIN	-TB	KC	CWS	-BAL	-NYY	-KC	BAL	NYY
CLE	NYY	BOS	TB	-BAL	LAD	SF	KC	-BOS	-KC	MIIN	-NYY	-MIIN
DET	MIIN	COL	-SD	CWS	-MIIN	-BOS	ARZ	MIIN	-TB	-NYY	KC	CWS
CWS	ARZ	SF	-MIIN	-DET	-KC	MIIN	-TOR	KC	MIIN	COL	-LAD	-DET
MIN	-DET	-SD	CWS	TOR	DET	-CWS	COL	-DET	-CWS	-CLE	BOS	CLE
KC	-SD	ARZ	-LAD	-SF	CWS	-TOR	-CLE	-CWS	CLE	TOR	-DET	-COL

Table A23 - Solution for DIV 4 Instance (- indicates home game) – Part I

TEAM	W13	W14	W15	W16	W17	W18	W19	W20	W21	W22	W23	W24
WAS	FLA	ATL	-FLA	-NYM	-COL	PHI	-ATL	NYY	BOS	-SF	-ARZ	-BOS
NYM	-BOS	PHI	TB	WAS	-TB	-ARZ	-NYY	FLA	ATL	-PHI	NYY	TOR
PHI	COL	-NYM	ATL	TB	-BAL	-WAS	-ARZ	BOS	-COL	NYM	-SF	FLA
ATL	TB	-WAS	-PHI	TOR	-BOS	-COL	WAS	-SD	-NYM	-TOR	LAD	SF
FLA	-WAS	-TB	WAS	BOS	TOR	-BOS	-SD	-NYM	ARZ	SD	COL	-PHI
ARZ	LAD	-SF	SD	SF	-LAD	NYM	PHI	CWS	-FLA	-MIIN	WAS	CLE
LAD	-ARZ	SD	-SF	-SD	ARZ	SF	-KC	CLE	DET	COL	-ATL	-DET
SF	SD	ARZ	LAD	-ARZ	-SD	-LAD	COL	-KC	CWS	WAS	PHI	-ATL
SD	-SF	-LAD	-ARZ	LAD	SF	-KC	FLA	ATL	CLE	-FLA	-DET	COL
COL	-PHI	-CLE	-KC	CLE	WAS	ATL	-SF	DET	PHI	-LAD	-FLA	-SD
BOS	NYM	NYY	-TOR	-FLA	ATL	FLA	TB	-PHI	-WAS	-NYY	TOR	WAS
NYY	BAL	-BOS	-CWS	-MIIN	KC	MIIN	NYM	-WAS	BAL	BOS	-NYM	-BAL
BAL	-NYY	-CWS	DET	CWS	PHI	-TB	MIIN	TB	-NYY	-CLE	KC	NYY
TB	-ATL	FLA	-NYM	-PHI	NYM	BAL	-BOS	-BAL	-TOR	KC	CWS	MIIN
TOR	-CWS	DET	BOS	-ATL	-FLA	CLE	-DET	MIIN	TB	ATL	-BOS	-NYM
CLE	KC	COL	MIIN	-COL	-DET	-TOR	CWS	-LAD	-SD	BAL	-MIIN	-ARZ
DET	-MIIN	-TOR	-BAL	KC	CLE	-CWS	TOR	-COL	-LAD	-CWS	SD	LAD
CWS	TOR	BAL	NYY	-BAL	-MIIN	DET	-CLE	-ARZ	-SF	DET	-TB	-KC
MIN	DET	-KC	-CLE	NYY	CWS	-NYY	-BAL	-TOR	KC	ARZ	CLE	-TB
KC	-CLE	MIIN	COL	-DET	-NYY	SD	LAD	SF	-MIIN	-TB	-BAL	CWS

Table A24 - Solution for DIV 4 Instance (- indicates home game) – Part II

TEAM	W25	W26	W27	W28	W29	W30	W31	W32	W33	W34	W35	W36
WAS	NYM	-TB	-BAL	TOR	-NYM	-TOR	NYM	-PHI	-NYY	BAL	-LAD	-PHI
NYM	-WAS	SD	ARZ	-PHI	WAS	-ATL	-WAS	ATL	FLA	-SF	-COL	-TOR
PHI	SD	SF	-FLA	NYM	-FLA	-NYY	-TB	WAS	TOR	-LAD	-TOR	WAS
ATL	ARZ	-FLA	-SF	FLA	NYY	NYM	-BAL	-NYM	-LAD	COL	SD	-ARZ
FLA	BAL	ATL	PHI	-ATL	PHI	-BAL	LAD	SF	-NYM	-ARZ	NYY	-COL
ARZ	-ATL	COL	-NYM	-SD	-LAD	COL	KC	MIIN	-CLE	FLA	DET	ATL
LAD	SF	-CWS	COL	MIIN	ARZ	-MIIN	-FLA	-COL	ATL	PHI	WAS	-SD
SF	-LAD	-PHI	ATL	DET	COL	-SD	-MIIN	-FLA	MIIN	NYM	CLE	-DET
SD	-PHI	-NYM	-CWS	ARZ	-MIIN	SF	-COL	-CLE	COL	CWS	-ATL	LAD
COL	MIIN	-ARZ	-LAD	CWS	-SF	-ARZ	SD	LAD	-SD	-ATL	NYM	FLA
BOS	-TOR	-NYY	TOR	BAL	TB	-CWS	-DET	-KC	CWS	KC	MIIN	-TB
NYY	-KC	BOS	-DET	-TB	-ATL	PHI	-TOR	CWS	WAS	TOR	-FLA	-BAL
BAL	-FLA	-MIIN	WAS	-BOS	-TOR	FLA	ATL	-DET	-KC	-WAS	TB	NYY
TB	-CWS	WAS	CLE	NYY	-BOS	-KC	PHI	TOR	-DET	-MIIN	-BAL	BOS
TOR	BOS	-CLE	-BOS	-WAS	BAL	WAS	NYY	-TB	-PHI	-NYY	PHI	NYM
CLE	-DET	TOR	-TB	-KC	CWS	DET	-CWS	SD	ARZ	DET	-SF	-CWS
DET	CLE	-KC	NYY	-SF	-KC	-CLE	BOS	BAL	TB	-CLE	-ARZ	SF
CWS	TB	LAD	SD	-COL	-CLE	BOS	CLE	-NYY	-BOS	-SD	KC	CLE
MIN	-COL	BAL	KC	-LAD	SD	LAD	SF	-ARZ	-SF	TB	-BOS	-KC
KC	NYY	DET	-MIIN	CLE	DET	TB	-ARZ	BOS	BAL	-BOS	-CWS	MIIN

Table A25 - Solution for DIV 4 Instance (- indicates home game) – Part III

TEAM	WAS	NYM	PHI	ATL	FLA	PIT	CHC	CIN	STL	HOU	MIL	ARZ	LAD	SF	SD
WAS	0	205	139	527	933	195	598	410	712	1233	640	1981	2308	2440	2275
NYM	205	0	66	732	1104	315	712	571	873	1433	734	2141	2457	2569	2431
PHI	139	66	0	666	1047	267	672	517	820	1368	700	2090	2409	2528	2381
ATL	527	732	666	0	617	509	579	369	468	727	660	1598	1950	2140	1895
FLA	933	1104	1047	617	0	1024	1195	972	1068	967	1275	1974	2339	2583	2258
PIT	195	315	267	509	1024	0	409	259	558	1152	446	1826	2143	2262	2116
CHC	598	712	672	579	1195	409	0	240	258	954	82	1452	1752	1857	1734
CIN	410	571	517	369	972	259	240	0	304	910	313	1574	1898	2033	1866
STL	712	873	820	468	1068	558	258	304	0	696	323	1270	1596	1741	1563
HOU	1233	1433	1368	727	967	1152	954	910	696	0	1017	1010	1374	1634	1291
MIL	640	734	700	660	1275	446	82	313	323	1017	0	1461	1750	1840	1738
ARZ	1981	2141	2090	1598	1974	1826	1452	1574	1270	1010	1461	0	365	645	298
LAD	2308	2457	2409	1950	2339	2143	1752	1898	1596	1374	1750	365	0	334	124
SF	2440	2569	2528	2140	2583	2262	1857	2033	1741	1634	1840	645	334	0	458
SD	2275	2431	2381	1895	2258	2116	1734	1866	1563	1291	1738	298	124	458	0
COL	1480	1616	1572	1200	1710	1305	906	1072	780	871	900	593	851	961	847
BOS	394	189	255	921	1270	480	848	739	1034	1617	856	2295	2600	2695	2581
NYY	205	0	66	732	1104	315	712	571	873	1433	734	2141	2457	2569	2431
BAL	30	176	109	557	960	200	607	428	731	1261	646	2001	2325	2454	2294
TB	803	992	929	418	207	865	997	782	862	807	1078	1801	2165	2398	2090
TOR	358	345	338	724	1249	226	437	409	660	1317	431	1888	2181	2267	2168
CLE	310	409	369	536	1094	115	303	211	484	1121	333	1741	2050	2160	2028
DET	408	496	461	575	1157	213	220	215	435	1102	239	1670	1973	2074	1954
CWS	598	712	672	579	1195	409	0	240	258	954	82	1452	1752	1857	1734
MIN	935	1016	988	900	1515	740	353	592	460	1067	296	1278	1531	1586	1535
KC	936	1096	1046	676	1241	781	414	535	237	654	442	1047	1364	1504	1335
OAK	2428	2556	2515	2127	2571	2249	1844	2021	1728	1623	1827	635	328	13	452
TEX	1178	1366	1305	727	1109	1063	794	807	541	242	846	885	1246	1474	1180
LAA	2276	2427	2379	1913	2297	2113	1724	1867	1564	1332	1724	324	44	374	89
SEA	2330	2407	2383	2182	2732	2136	1737	1961	1721	1891	1691	1111	958	687	1071

Table A26 - Table Distance Matrix for Actual MLB Schedule Instance – PART I

	COL	BOS	NYY	BAL	TB	TOR	CLE	DET	CWS	MIN	KC	OAK	TEX	LAA	SEA
WAS	1480	394	205	30	803	358	310	408	598	935	936	2428	1178	2276	2330
NYM	1616	189	0	176	992	345	409	496	712	1016	1096	2556	1366	2427	2407
PHI	1572	255	66	109	929	338	369	461	672	988	1046	2515	1305	2379	2383
ATL	1200	921	732	557	418	724	536	575	579	900	676	2127	727	1913	2182
FLA	1710	1270	1104	960	207	1249	1094	1157	1195	1515	1241	2571	1109	2297	2732
PIT	1305	480	315	200	865	226	115	213	409	740	781	2249	1063	2113	2136
CHC	906	848	712	607	997	437	303	220	0	353	414	1844	794	1724	1737
CIN	1072	739	571	428	782	409	211	215	240	592	535	2021	807	1867	1961
STL	780	1034	873	731	862	660	484	435	258	460	237	1728	541	1564	1721
HOU	871	1617	1433	1261	807	1317	1121	1102	954	1067	654	1623	242	1332	1891
MIL	900	856	734	646	1078	431	333	239	82	296	442	1827	846	1724	1691
ARZ	593	2295	2141	2001	1801	1888	1741	1670	1452	1278	1047	635	885	324	1111
LAD	851	2600	2457	2325	2165	2181	2050	1973	1752	1531	1364	328	1246	44	958
SF	961	2695	2569	2454	2398	2267	2160	2074	1857	1586	1504	13	1474	374	687
SD	847	2581	2431	2294	2090	2168	2028	1954	1734	1535	1335	452	1180	89	1071
COL	0	1753	1616	1495	1510	1331	1207	1126	906	689	544	949	643	827	1033
BOS	1753	0	189	364	1170	430	554	628	848	1119	1248	2682	1542	2572	2489
NYY	1616	189	0	176	992	345	409	496	712	1016	1096	2556	1366	2427	2407
BAL	1495	364	176	0	832	344	314	411	607	939	962	2442	1203	2294	2336
TB	1510	1170	992	832	0	1091	922	977	997	1311	1034	2386	924	2124	2529
TOR	1331	430	345	344	1091	0	199	226	437	689	849	2254	1196	2155	2069
CLE	1207	554	409	314	922	199	0	98	303	626	694	2148	1010	2021	2022
DET	1126	628	496	411	977	226	98	0	220	529	627	2062	973	1945	1926
CWS	906	848	712	607	997	437	303	220	0	353	414	1844	794	1724	1737
MIN	689	1119	1016	939	1311	689	626	529	353	0	413	1574	851	1510	1397
KC	544	1248	1096	962	1034	849	694	627	414	413	0	1491	440	1333	1507
OAK	949	2682	2556	2442	2386	2254	2148	2062	1844	1574	1491	0	1462	367	684
TEX	643	1542	1366	1203	924	1196	1010	973	794	851	440	1462	0	1207	1673
LAA	827	2572	2427	2294	2124	2155	2021	1945	1724	1510	1333	367	1207	0	984
SEA	1033	2489	2407	2336	2529	2069	2022	1926	1737	1397	1507	684	1673	984	0

Table A27 - Table Distance Matrix for Actual MLB Schedule Instance – PART II

H/A	WAS	NYM	PHI	ATL	FLA	PIT	CHC	CIN	STL	HOU	MIL	ARZ	LAD	SF	SD
WAS	0	3	3	3	3	1	1	1	1	1	1	1	1	1	1
NYM	3	0	3	3	3	1	1	1	1	1	1	1	1	1	1
PHI	3	3	0	3	3	1	1	1	1	1	1	1	1	1	1
ATL	3	3	3	0	3	1	1	1	1	1	1	1	1	1	1
FLA	3	3	3	3	0	1	1	1	1	1	1	1	1	1	2
PIT	1	1	1	2	1	0	2	3	3	2	3	1	1	1	1
CHC	1	1	1	1	1	3	0	2	3	3	2	1	1	1	1
CIN	1	1	1	1	1	2	3	0	3	2	3	1	2	1	1
STL	1	1	1	1	1	2	2	3	0	3	3	1	1	1	1
HOU	1	1	1	1	1	3	3	3	2	0	2	1	1	2	1
MIL	1	1	1	1	1	3	3	2	2	3	0	1	1	1	1
ARZ	1	1	1	1	1	1	1	1	2	1	1	0	3	3	3
LAD	1	1	1	1	1	1	1	1	1	1	1	3	0	3	3
SF	1	1	1	1	1	1	1	1	1	1	1	3	3	0	3
SD	1	1	1	1	1	1	1	1	1	1	1	3	3	3	0
COL	1	1	1	1	1	1	1	1	1	1	2	3	3	3	3
BOS	0	0	1	0	0	0	0	0	0	0	0	1	1	0	0
NYY	0	1	1	0	0	0	0	0	0	1	0	0	0	0	0
BAL	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0
TB	0	0	0	0	1	0	0	0	0	0	0	1	0	0	1
TOR	0	0	1	0	0	0	0	0	1	0	0	0	0	1	0
CLE	1	1	0	0	0	0	0	1	0	0	0	0	0	0	0
DET	1	0	0	0	0	1	0	0	0	0	0	1	0	0	0
CWS	0	0	0	1	1	0	1	0	0	0	0	0	0	0	0
MIN	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0
KC	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0
OAK	0	0	0	0	0	1	0	1	0	0	0	0	0	1	0
TEX	0	0	0	0	0	1	1	0	0	1	0	0	0	0	0
LAA	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0
SEA	0	0	0	0	0	0	1	1	0	0	0	0	0	0	1

Table A28 - Match-Ups for Actual MLB Instance – Part I

	COL	BOS	NYY	BAL	TB	TOR	CLE	DET	CWS	MIN	KC	OAK	TEX	LAA	SEA
WAS	1	0	0	1	0	0	0	0	1	0	1	0	0	0	0
NYM	1	0	1	0	0	0	0	1	0	1	0	0	0	0	0
PHI	1	1	0	0	0	0	1	0	0	1	0	0	0	0	0
ATL	1	0	0	0	1	0	0	1	0	0	1	0	0	0	0
FLA	1	0	0	0	1	0	0	0	0	0	0	0	1	0	0
PIT	1	0	0	0	0	0	1	0	1	0	0	0	0	0	0
CHC	1	0	0	0	0	0	0	0	1	0	0	1	0	1	0
CIN	1	0	0	0	0	0	1	0	0	0	1	0	0	0	0
STL	1	0	0	0	0	0	0	0	0	0	0	1	0	1	1
HOU	1	0	0	0	1	0	0	0	0	0	0	0	1	0	0
MIL	1	0	0	0	0	0	0	0	0	1	0	0	1	0	1
ARZ	3	0	1	0	0	1	0	0	0	0	0	0	0	0	0
LAD	3	0	1	0	0	0	0	1	0	0	0	0	0	1	0
SF	3	1	0	1	0	0	0	0	0	0	0	1	0	0	0
SD	3	0	0	1	0	1	0	0	0	0	0	0	0	0	1
COL	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0
BOS	0	0	3	3	3	3	1	1	1	1	1	1	2	2	1
NYY	0	3	0	3	3	3	1	1	1	1	1	1	1	2	2
BAL	0	3	3	0	3	3	1	1	1	1	1	2	1	1	2
TB	0	3	3	3	0	3	2	1	1	1	1	1	1	1	2
TOR	0	3	3	3	3	0	1	1	1	2	1	1	2	1	1
CLE	0	1	1	1	1	2	0	3	3	3	3	2	1	1	1
DET	0	1	1	2	1	1	3	0	3	3	3	1	1	2	1
CWS	0	1	1	1	1	1	3	3	0	3	3	1	1	2	2
MIN	1	1	1	1	1	1	3	3	3	0	3	2	2	1	1
KC	1	1	1	1	1	1	3	3	3	3	0	2	2	1	1
OAK	0	2	2	1	2	1	1	1	2	1	1	0	3	3	3
TEX	0	1	2	2	1	1	1	2	2	1	1	3	0	3	3
LAA	1	1	1	1	2	2	2	1	1	1	2	3	3	0	3

SEA	0	2	1	1	1	1	1	2	1	2	2	3	3	3	0

Table A29 - Match-Ups for Actual MLB Instance – Part II

TEAM	**WAS**	**NYM**	**PHI**	**ATL**	**FLA**
Tile1	SD,LAD,SF	SD,LAD,SF	SD,LAD,SF	FLA,NYM,PHI	SD,LAD,SF
Tile2	STL,COL,ARZ	STL,COL,ARZ	STL,COL,ARZ	SD,LAD,SF	BAL,PHI,NYM
Tile3	CHC,MIL,HOU	CHC,MIL,HOU	CHC,MIL,HOU	MIN,COL,ARZ	WAS,PHI,NYM
Tile4	CIN,ATL,FLA	CIN,ATL,FLA	CIN,ATL,FLA	WAS,PHI,NYM	PIT,PHI,NYM
Tile5	DET,ATL,FLA	CLE,ATL,FLA	PIT,ATL,FLA	CHC,CWS,MIL	HOU,COL,ARZ
Tile6	CLE,ATL,FLA	PIT,ATL,FLA	TOR,ATL,FLA	CIN,STL,HOU	ATL,CIN,STL
Tile7	PIT,PHI,NYM	PHI,BAL,WAS	NYM,NYY,WAS	FLA,PIT,WAS	CHC,CWS,MIL
Tile8	BAL,PHI,NYM	PHI,WAS	NYM,BOS,WAS	FLA,WAS,PHI	TB,ATL,WAS
Tile9	NYM,PHI	NYY,PHI,WAS	NYM,WAS	NYM,PIT	ATL,WAS

Table A30 - Tiles for Actual MLB Instance Part I

TEAM	**PIT**	**CHC**	**CIN**	**STL**	**HOU**	**MIL**
Tile1	MIL,HOU	LAD,SF,SEA	OAK,SF,SEA	CHC,CIN,PIT	TEX,ATL,FLA	WAS,PHI,NYM
Tile2	LAD,OAK,SF	COL,ARZ,SD	ARZ,SD,LAD	SD,LAD,SF	PHI,NYM,NYY	LAA,LAD,SF
Tile3	COL,ARZ,SD	WAS,PHI,NYM	CHC,CLE,PIT	KC,COL,ARZ	CIN,PIT,WAS	COL,ARZ,SD
Tile4	STL,TEX,HOU	CIN,ATL,FLA	WAS,PHI,NYM	HOU,CHC,MIL	STL,CHC,MIL	CHC,CIN,PIT
Tile5	CIN,CHC,MIL	CWS,MIL,HOU	STL,HOU,COL	WAS,PHI,NYM	SD,SF,LAD	MIN,ATL,FLA
Tile6	CIN,ATL,FLA	CIN,STL,TEX	MIL,ATL,FLA	CIN,ATL,FLA	KC,COL,ARZ	PIT,STL,HOU
Tile7	PHI,NYM,DET	HOU,MIL	PIT,STL,HOU	CIN,PIT,TOR	STL,CHC,MIL	CHC,CIN,PIT
Tile8	WAS,CHC,STL	STL,CIN,PIT	CHC,STL,HOU	MIL,CHC,PIT	STL,CHC,MIL	STL,CIN,COL
Tile9	CHC,MIL,HOU	PIT,HOU,MIL	PIT,MIL	HOU,ARZ	CIN,PIT	HOU,STL

Table A31 - Tiles for Actual MLB Instance Part II

TEAM	ARZ	LAD	SF	SD	COL
Tile1	ATL,TB,FLA	PHI,NYM,BOS	WAS,PHI,NYM	WAS,PHI,NYM	WAS,PHI,NYM
Tile2	PHI,NYM,BOS	CIN,ATL,FLA	CIN,PIT,TOR	ATL,TB,FLA	CHC,MIL,MIN
Tile3	DET,PIT,WAS	CHC,PIT,WAS	CHC,ATL,FLA	CHC,CIN,PIT	STL,CIN,PIT
Tile4	CHC,MIL,CIN	STL,MIL,HOU	STL,HOU,MIL	STL,MIL,HOU	LAA,LAD,SF
Tile5	SD,LAD,SF	SD,ARZ,COL	SD,ARZ,LAD	LAD,SF,SEA	ARZ,SD,SF
Tile6	COL,STL,HOU	LAA,SF,SD	OAK,LAD,SD	ARZ,COL	SD,LAD,SF
Tile7	SD,LAD,SF	ARZ,SD,COL	ARZ,COL,HOU	LAD,SF,ARZ	ARZ,KC,HOU
Tile8	SD,COL,LAD	SF,ARZ,CIN	SD,ARZ,COL	LAD,SF,COL	ARZ,ATL,FLA
Tile9	SF,COL	COL,SF	LAD,COL	ARZ,COL,FLA	LAD,SD

Table A32 - Tiles for Actual MLB Instance Part III

TEAM	BOS	NYY	BAL	TB	TOR
Tile1	OAK,SF,SEA	LAD,OAK,SEA	BOS,TB,TOR	LAA,OAK,SEA	LAA,OAK,SEA
Tile2	KC,COL,LAA	TEX,ARZ,LAA	OAK,SF,SEA	KC,TEX,HOU	COL,ARZ,SD
Tile3	CWS,MIN,TEX	CWS,MIN,KC	TEX,SD,LAA	DET,CWS,MIN	MIN,KC,TEX
Tile4	CLE,DET,TOR	CLE,DET,TOR	CWS,MIN,KC	BAL,NYY,BOS	CLE,DET,CWS
Tile5	PHI,BAL,TB	NYM,BOS,TB	CLE,DET,TOR	ATL,CLE,BOS	BAL,NYY,BOS
Tile6	NYY,BAL,TB	BOS,BAL,TB	WAS,TB,NYY	FLA,BAL,TOR	NYY,BOS,TB
Tile7	NYY,BAL,TOR	BAL,TB,TOR	NYY,BOS,TB	TOR,BAL,NYY	BOS,TB,BAL
Tile8	NYY,TB,TOR	BOS,BAL,TOR	NYY,BOS,TOR	TOR,NYY,BOS	BAL,TB,NYY
Tile9	OAK,SEA	OAK,TEX	DET,TEX	OAK,LAA	LAA,CLE

Table A33 - Tiles for Actual MLB Instance Part IV

TEAM	CLE	DET	CWS	MIN	KC
Tile1	LAA,SEA,OAK	TEX,SEA	OAK,TEX	TOR,SEA	LAA,SEA
Tile2	MIN,KC,TEX	LAD,OAK,SEA	BAL,NYY,BOS	NYM,BOS,NYY	LAA,OAK,SEA
Tile3	PHI,NYY,BOS	KC,TEX,LAA	PIT,WAS,TB	BAL,PHI,TB	BAL,NYY,BOS
Tile4	TOR,DET,CWS	NYM,NYY,BOS	DET,CLE,TOR	DET,CLE,TOR	CIN,ATL,TB
Tile5	PIT,BAL	CLE,ATL,TB	LAA,OAK,SEA	MIL,CWS,CLE	DET,CLE,TOR
Tile6	KC,CIN,TB	CWS,MIN,KC	MIN,KC,TEX	KC,DET,TEX	CWS,DET,CLE
Tile7	DET,CWS,MIN	TOR,CLE,BAL	CHC,DET,CLE	LAA,OAK,SEA	MIN,CWS,DET
Tile8	CWS,MIN,KC	CWS,MIN,KC	KC,CLE,MIN	DET,KC,CWS	MIN,CLE,CWS
Tile9	TB,LAA,DET	CLE,CWS,MIN	KC,DET,MIN	CLE,CWS,KC	MIN,WAS,TEX

Table A34 - Tiles for Actual MLB Instance Part V

TEAM	OAK	TEX	LAA	SEA
Tile1	CLE,BAL	LAA,OAK,SEA	TEX,OAK,SEA	OAK,LAA,TEX
Tile2	BAL,NYY,BOS	LAA,OAK,SEA	TEX,OAK,SEA	OAK,LAA,TEX
Tile3	CLE,TOR,TB	LAA,OAK,SEA	TEX,OAK,SEA	OAK,LAA,TEX
Tile4	KC,STL,CHC	BAL,NYY,BOS	TEX,BAL,NYY,BOS	TB,NYY,BOS
Tile5	MIN,CWS,DET	DET,CLE,TOR	CLE,TOR,TB	CLE,TOR,BAL
Tile6	LAA,SF,SEA	MIN,MIL,CWS	MIN,CWS,DET	CWS,MIL,MIN
Tile7	LAA,TEX,SEA	HOU,TB,FLA	KC,STL,CHC	KC,STL,DET
Tile8	LAA,TEX,SEA	KC,BOS,TOR	LAD,BOS,NYY	SD,NYY,BAL
Tile9	TEX,KC,MIN	KC,MIN	DET,CWS	TB,CWS

Table A35 - Tiles for Actual MLB Instance Part VI

TEAM	W1	W2	W3	W4	W5	W6	W7	W8	W9	W10	W11	W12	W13
WAS	FLA	NYM	-FLA	-NYM	-COL	NYM	-PHI	CIN	-ATL	-FLA	NYM	PHI	STL
NYM	CLE	-WAS	-PIT	WAS	CHC	-WAS	-SF	PHI	-CHC	-ATL	-WAS	FLA	ATL
PHI	-LAD	NYY	CIN	-COL	-FLA	-MIL	WAS	-NYM	-CLE	COL	-ATL	-WAS	-PIT
ATL	HOU	STL	-SD	-FLA	ARZ	SF	LAD	-HOU	WAS	NYM	PHI	-ARZ	-NYM
FLA	-WAS	-SD	WAS	ATL	PHI	-LAD	-CIN	COL	PIT	WAS	-TEX	-NYM	-ARZ
PIT	-SD	-CHC	NYM	-CIN	-MIL	ARZ	SD	OAK	-FLA	-STL	-CIN	CHC	PHI
CHC	SF	PIT	-STL	-HOU	-NYM	HOU	MIL	-ARZ	NYM	CIN	-OAK	-PIT	-COL
CIN	STL	-LAD	-PHI	PIT	-LAD	-STL	FLA	-WAS	HOU	-CHC	PIT	-STL	-MIL
STL	-CIN	-ATL	CHC	MIL	-HOU	CIN	-COL	-SF	TOR	PIT	-LAD	CIN	-WAS
HOU	-ATL	-COL	-MIL	CHC	STL	-CHC	-ARZ	ATL	-CIN	-SD	COL	SF	LAD
MIL	LAA	ARZ	HOU	-STL	PIT	PHI	-CHC	SD	LAD	SF	-ARZ	-COL	CIN
ARZ	-COL	-MIL	SF	SD	-ATL	-PIT	HOU	CHC	-SF	-LAD	MIL	ATL	FLA
LAD	PHI	CIN	COL	-SF	CIN	FLA	-ATL	-NYY	-MIL	ARZ	STL	SD	-HOU
SF	-CHC	-BAL	-ARZ	LAD	SD	-ATL	NYM	STL	ARZ	-MIL	-SD	-HOU	SD
SD	PIT	FLA	ATL	-ARZ	-SF	COL	-PIT	-MIL	-COL	HOU	SF	-LAD	-SF
COL	ARZ	HOU	-LAD	PHI	WAS	-SD	STL	-FLA	SD	-PHI	-HOU	MIL	CHC
BOS	BAL	-TOR	NYY	-BAL	-TEX	-TB	TOR	-KC	-LAA	-NYY	TOR	DET	NYY
NYY	TOR	-PHI	-BOS	-TEX	OAK	-LAA	TEX	LAD	-BAL	BOS	-DET	-MIN	-BOS
BAL	-BOS	SF	OAK	BOS	-SEA	-TOR	-KC	TB	NYY	-DET	-SEA	CLE	-TB
TB	TEX	OAK	-MIN	-TOR	CWS	BOS	CLE	-BAL	SEA	KC	LAA	-SEA	BAL
TOR	-NYY	BOS	-CWS	TB	KC	BAL	-BOS	-LAA	-STL	CWS	-BOS	-TEX	-MIN
CLE	-NYM	KC	TEX	-CWS	MIN	-SEA	-TB	-MIN	PHI	MIN	KC	-BAL	CWS
DET	CWS	MIN	-LAA	SEA	LAA	KC	-MIN	-CWS	MIN	BAL	NYY	-BOS	-KC
CWS	-DET	-LAA	TOR	CLE	-TB	-MIN	-SEA	DET	-KC	-TOR	MIN	KC	-CLE
MIN	KC	-DET	TB	-KC	-CLE	CWS	DET	CLE	-DET	-CLE	-CWS	NYY	TOR
KC	-MIN	-CLE	-SEA	MIN	-TOR	-DET	BAL	BOS	CWS	-TB	-CLE	-CWS	DET
OAK	-SEA	-TB	-BAL	LAA	-NYY	-TEX	LAA	-PIT	-TEX	SEA	CHC	LAA	-SEA
TEX	-TB	-SEA	-CLE	NYY	BOS	OAK	-NYY	SEA	OAK	-LAA	FLA	TOR	LAA
LAA	-MIL	CWS	DET	-OAK	-DET	NYY	-OAK	TOR	BOS	TEX	-TB	-OAK	-TEX
SEA	OAK	TEX	KC	-DET	BAL	CLE	CWS	-TEX	-TB	-OAK	BAL	TB	OAK

Table A36 - Tiling Solution for Actual MLB Instance (- indicates home game) – Part I

TEAM	W14	W15	W16	W17	W18	W19	W20	W21	W22	W23	W24	W25	W26
WAS	-MIL	-PIT	HOU	SD	SF	-CIN	CHC	CLE	PHI	-SD	-KC	-BAL	ATL
NYM	-PHI	COL	SF	MIL	-PHI	-FLA	-SD	BAL	CIN	ATL	-DET	-PHI	FLA
PHI	NYM	-BOS	ATL	FLA	NYM	-SD	-ATL	BOS	-WAS	-MIN	-STL	NYM	TOR
ATL	PIT	-CIN	-PHI	-TB	FLA	-DET	PHI	MIN	SD	-NYM	CWS	CHC	-WAS
FLA	STL	CWS	CIN	-PHI	-ATL	NYM	ARZ	TB	-HOU	-TB	BAL	-SD	-NYM
PIT	-ATL	WAS	MIL	-CWS	-HOU	-LAD	MIL	DET	-ARZ	-STL	HOU	LAD	CHC
CHC	TEX	HOU	-LAD	STL	MIL	CWS	-WAS	-CWS	-SF	COL	SEA	-ATL	-PIT
CIN	HOU	ATL	-FLA	-HOU	STL	WAS	-HOU	-KC	-NYM	SEA	OAK	-STL	CLE
STL	-FLA	-LAA	COL	-CHC	-CIN	-MIL	SF	ARZ	-MIL	PIT	PHI	CIN	-ARZ
HOU	-CIN	-CHC	-WAS	CIN	PIT	-SF	CIN	NYY	FLA	-MIL	-PIT	-TEX	MIL
MIL	WAS	MIN	-PIT	-NYM	-CHC	STL	-PIT	-TEX	STL	HOU	-MIN	-SEA	-HOU
ARZ	-SF	-TOR	-SD	LAD	SD	-COL	-FLA	-STL	PIT	DET	-NYY	TB	STL
LAD	-SD	-DET	CHC	-ARZ	-COL	PIT	COL	LAA	-COL	BOS	-LAA	-PIT	SF
SF	ARZ	OAK	-NYM	-COL	-WAS	HOU	-STL	-OAK	CHC	TOR	-COL	-BOS	-LAD
SD	LAD	SEA	ARZ	-WAS	-ARZ	PHI	NYM	-SEA	-ATL	WAS	TB	FLA	-COL
COL	KC	-NYM	-STL	SF	LAD	ARZ	-LAD	-TOR	LAD	-CHC	SF	LAA	SD
BOS	-MIN	PHI	TB	KC	-CLE	BAL	CLE	-PHI	-OAK	-LAD	TEX	SF	TB
NYY	BAL	TB	MIN	-CLE	-CWS	-TB	BAL	-HOU	-BAL	LAA	ARZ	-OAK	-SEA
BAL	-NYY	TEX	-CWS	DET	TEX	-BOS	-NYY	-NYM	NYY	-OAK	-FLA	WAS	-OAK
TB	-CLE	-NYY	-BOS	ATL	TOR	NYY	-TOR	-FLA	-KC	FLA	-SD	-ARZ	-BOS
TOR	SEA	ARZ	LAA	-OAK	-TB	-CLE	TB	COL	MIN	-SF	CLE	-MIN	-PHI
CLE	TB	-KC	-OAK	NYY	BOS	TOR	-BOS	-WAS	DET	-CWS	-TOR	CWS	-CIN
DET	OAK	LAD	SEA	-BAL	-OAK	ATL	CWS	-PIT	-CLE	-ARZ	NYM	-KC	MIN
CWS	-LAA	-FLA	BAL	PIT	NYY	-CHC	-DET	CHC	-TEX	CLE	-ATL	-CLE	KC
MIN	BOS	-MIL	-NYY	-TEX	SEA	OAK	KC	-ATL	-TOR	PHI	MIL	TOR	-DET
KC	-COL	CLE	-TEX	-BOS	LAA	TEX	-MIN	CIN	TB	-TEX	WAS	DET	-CWS
OAK	-DET	-SF	CLE	TOR	DET	-MIN	-LAA	SF	BOS	BAL	-CIN	NYY	BAL
TEX	-CHC	-BAL	KC	MIN	-BAL	-KC	-SEA	MIL	CWS	KC	-BOS	HOU	LAA
LAA	CWS	STL	-TOR	-SEA	-KC	SEA	OAK	-LAD	-SEA	-NYY	LAD	-COL	-TEX
SEA	-TOR	-SD	-DET	LAA	-MIN	-LAA	TEX	SD	LAA	-CIN	-CHC	MIL	NYY

Table A37 - Tiling Solution for Actual MLB Instance (- indicates home game) – Part II

TEAM	W27	W28	W29	W30	W31	W32	W33	W34	W35	W36	W37	W38	W39
WAS	-NYM	-CHC	-SF	FLA	ATL	MIL	-ATL	COL	LAD	ARZ	-PHI	-NYM	PIT
NYM	WAS	-ATL	-CIN	SD	ARZ	LAD	-MIN	-ARZ	-ATL	PHI	-MIL	WAS	HOU
PHI	-CHC	-CIN	-ATL	CHC	STL	FLA	-ARZ	MIL	-FLA	-NYM	WAS	ATL	-SF
ATL	FLA	NYM	PHI	-MIL	-WAS	-COL	WAS	CIN	NYM	-SF	-FLA	-PHI	FLA
FLA	-ATL	LAD	HOU	-WAS	-COL	-PHI	SD	SF	PHI	-STL	ATL	-MIL	-ATL
PIT	-SF	HOU	CHC	-HOU	-MIL	SF	COL	STL	-CIN	-COL	TEX	CIN	-WAS
CHC	PHI	WAS	-PIT	-PHI	-HOU	STL	HOU	-LAA	-MIL	-CIN	SD	LAD	-SD
CIN	-MIL	PHI	NYM	-COL	-SD	-CLE	MIL	-ATL	PIT	CHC	STL	-PIT	ARZ
STL	-HOU	ARZ	SD	LAD	-PHI	-CHC	KC	-PIT	HOU	FLA	-CIN	HOU	-MIL
HOU	STL	-PIT	-FLA	PIT	CHC	-TB	-CHC	TEX	-STL	MIL	-SF	-STL	-NYM
MIL	CIN	-SF	COL	ATL	PIT	-WAS	-CIN	-PHI	CHC	-HOU	NYM	FLA	STL
ARZ	COL	-STL	LAD	SF	-NYM	-SD	PHI	NYM	SD	-WAS	-LAD	COL	-CIN
LAD	SD	-FLA	-ARZ	-STL	SF	-NYM	-SF	SD	-WAS	-SD	ARZ	-CHC	-COL
SF	PIT	MIL	WAS	-ARZ	-LAD	-PIT	LAD	-FLA	COL	ATL	HOU	-SD	PHI
SD	-LAD	COL	-STL	-NYM	CIN	ARZ	-FLA	-LAD	-ARZ	LAD	-CHC	SF	CHC
COL	-ARZ	-SD	-MIL	CIN	FLA	ATL	-PIT	-WAS	-SF	PIT	-BOS	-ARZ	LAD
BOS	-BAL	-TB	TOR	-TEX	OAK	SEA	LAA	-DET	-TOR	MIN	COL	CWS	-NYY
NYY	-TOR	OAK	SEA	KC	-BAL	-KC	CLE	TB	-LAA	TOR	DET	-TB	BOS
BAL	BOS	-TOR	DET	TOR	NYY	-MIN	TOR	CWS	MIN	-LAA	-CLE	KC	TB
TB	MIN	BOS	-TEX	-OAK	-LAA	HOU	-DET	-NYY	-CLE	DET	TOR	NYY	-BAL
TOR	NYY	BAL	-BOS	-BAL	-KC	DET	-BAL	CLE	BOS	-NYY	-TB	LAA	OAK
CLE	-OAK	DET	-MIN	-DET	-TEX	CIN	-NYY	-TOR	TB	-CWS	BAL	-DET	KC
DET	-SEA	-CLE	-BAL	CLE	-MIN	-TOR	TB	BOS	-CWS	-TB	-NYY	CLE	-TEX
CWS	TEX	-MIN	-KC	MIN	SEA	OAK	-SEA	-BAL	DET	CLE	-MIN	-BOS	MIN
MIN	-TB	CWS	CLE	-CWS	DET	BAL	NYM	-SEA	-BAL	-BOS	CWS	-OAK	-CWS
KC	-LAA	SEA	CWS	-NYY	TOR	NYY	-STL	-OAK	SEA	OAK	LAA	-BAL	-CLE
OAK	CLE	-NYY	-LAA	TB	-BOS	-CWS	TEX	KC	-TEX	-KC	SEA	MIN	-TOR
TEX	-CWS	-LAA	TB	BOS	CLE	-LAA	-OAK	-HOU	OAK	SEA	-PIT	-SEA	DET
LAA	KC	TEX	OAK	-SEA	TB	TEX	-BOS	CHC	NYY	BAL	-KC	-TOR	SEA
SEA	DET	-KC	-NYY	LAA	-CWS	-BOS	CWS	MIN	-KC	-TEX	-OAK	TEX	-LAA

Table A38 - Tiling Solution for Actual MLB Instance (- indicates home game) – Part III

TEAM	W40	W41	W42	W43	W44	W45	W46	W47	W48	W49	W50	W51	W52
WAS	-PHI	-CWS	-STL	FLA	ATL	BAL	-FLA	-HOU	-ATL	PHI	-LAD	DET	-ARZ
NYM	PIT	-FLA	-HOU	NYY	-FLA	-NYY	PHI	-COL	STL	FLA	ATL	-LAD	-STL
PHI	WAS	-HOU	SD	LAD	PIT	-FLA	-NYM	ATL	FLA	-WAS	ARZ	SF	HOU
ATL	-CHC	COL	-FLA	-KC	-WAS	MIL	-STL	-PHI	WAS	PIT	-NYM	-PIT	-LAD
FLA	-SF	NYM	ATL	-WAS	NYM	PHI	WAS	-CHC	-PHI	-NYM	MIL	CHC	-PIT
PIT	-NYM	-CLE	MIL	-STL	-PHI	STL	CIN	-MIL	-CHC	-ATL	HOU	ATL	FLA
CHC	ATL	ARZ	CIN	-MIL	-STL	-CIN	MIL	FLA	PIT	-HOU	-STL	-FLA	CIN
CIN	LAD	-SF	-CHC	HOU	MIL	CHC	-PIT	-ARZ	SF	COL	SD	-MIL	-CHC
STL	-HOU	-OAK	WAS	PIT	CHC	-PIT	ATL	-SD	-NYM	MIL	CHC	-SEA	NYM
HOU	STL	PHI	NYM	-CIN	ARZ	SD	-LAD	WAS	MIL	CHC	-PIT	KC	-PHI
MIL	-SD	-LAD	-PIT	CHC	-CIN	-ATL	-CHC	PIT	-HOU	-STL	-FLA	CIN	COL
ARZ	-COL	-CHC	SF	-SD	-HOU	-LAD	COL	CIN	LAD	-SF	-PHI	BOS	WAS
LAD	-CIN	MIL	COL	-PHI	SF	ARZ	HOU	-SF	-ARZ	-SD	WAS	NYM	ATL
SF	FLA	CIN	-ARZ	-COL	-LAD	COL	SD	LAD	-CIN	ARZ	COL	-PHI	-SD
SD	MIL	-BAL	-PHI	ARZ	-COL	-HOU	-SF	STL	COL	LAD	-CIN	-TOR	SF
COL	ARZ	-ATL	-LAD	SF	SD	-SF	-ARZ	NYM	-SD	-CIN	-SF	MIN	-MIL
BOS	-TOR	-SEA	TB	BAL	-CWS	-TB	OAK	SEA	-LAA	-BAL	NYY	-ARZ	-NYY
NYY	-SEA	TOR	CWS	-NYM	-TOR	NYM	TEX	TB	-TOR	-TB	-BOS	BAL	BOS
BAL	-TEX	SD	LAA	-BOS	-TB	-WAS	SEA	TOR	-TB	BOS	-TOR	-NYY	TB
TB	OAK	LAA	-BOS	-TOR	BAL	BOS	TOR	-NYY	BAL	NYY	-SEA	-CWS	-BAL
TOR	BOS	-NYY	-DET	TB	NYY	-TEX	-TB	-BAL	NYY	-SEA	BAL	SD	TEX
CLE	DET	PIT	-KC	CWS	SEA	LAA	-MIN	-LAA	-KC	MIN	OAK	LAA	-DET
DET	-CLE	-KC	TOR	TEX	KC	-CWS	-LAA	-MIN	CWS	KC	TEX	-WAS	CLE
CWS	KC	WAS	-NYY	-CLE	BOS	DET	-KC	TEX	-DET	OAK	LAA	TB	-OAK
MIN	-LAA	TEX	SEA	LAA	-TEX	-KC	CLE	DET	-OAK	-CLE	KC	-COL	-KC
KC	-CWS	DET	CLE	ATL	-DET	MIN	CWS	-OAK	CLE	-DET	-MIN	-HOU	MIN
OAK	-TB	STL	TEX	SEA	-LAA	-SEA	-BOS	KC	MIN	-CWS	-CLE	TEX	CWS
TEX	BAL	-MIN	-OAK	-DET	MIN	TOR	-NYY	-CWS	SEA	LAA	-DET	-OAK	-TOR
LAA	MIN	-TB	-BAL	-MIN	OAK	-CLE	DET	CLE	BOS	-TEX	-CWS	-CLE	SEA
SEA	NYY	BOS	-MIN	-OAK	-CLE	OAK	-BAL	-BOS	-TEX	TOR	TB	STL	-LAA

Table A39 - Tiling Solution for Actual MLB Instance (- indicates home game) – Part IV

TEAM	W1	W2	W3	W4	W5	W6	W7	W8	W9	W10	W11	W12	W13
WAS	-PHI	NYM	PHI	-MIL	-COL	-LAD	CHC	FLA	-ATL	-FLA	NYM	COL	STL
NYM	-FLA	-WAS	COL	STL	-CHC	-ATL	-LAD	PHI	CIN	-SF	-WAS	FLA	ATL
PHI	WAS	HOU	-WAS	-FLA	ATL	ARZ	SF	-NYM	-STL	-ATL	COL	MIL	-PIT
ATL	-CHC	SF	SD	-COL	-PHI	NYM	STL	-HOU	WAS	PHI	MIL	-ARZ	-NYM
FLA	NYM	-LAD	-CIN	PHI	HOU	COL	-SD	-WAS	-SF	WAS	CHC	-NYM	-ARZ
PIT	-LAD	ARZ	SF	-CIN	-MIL	HOU	MIL	LAD	-CHC	-STL	-CIN	CHC	PHI
CHC	ATL	CIN	-MIL	-HOU	NYM	MIL	-WAS	-ARZ	PIT	CIN	-FLA	-PIT	-COL
CIN	-STL	-CHC	FLA	PIT	-LAD	-SD	HOU	STL	-NYM	-CHC	PIT	-STL	-MIL
STL	CIN	MIL	-HOU	-NYM	ARZ	SF	-ATL	-CIN	PHI	PIT	-HOU	CIN	-WAS
HOU	-SF	-PHI	STL	CHC	-FLA	-PIT	-CIN	ATL	-ARZ	-SD	STL	SF	LAD
MIL	-COL	-STL	CHC	WAS	PIT	-CHC	-PIT	SD	LAD	ARZ	-ATL	-PHI	CIN
ARZ	-SD	-PIT	LAD	SD	-STL	-PHI	COL	CHC	HOU	-MIL	-LAD	ATL	FLA
LAD	PIT	FLA	-ARZ	-SF	CIN	WAS	NYM	-PIT	-MIL	-COL	ARZ	SD	-HOU
SF	HOU	-ATL	-PIT	LAD	SD	-STL	-PHI	-COL	FLA	NYM	-SD	-HOU	SD
SD	ARZ	COL	-ATL	-ARZ	-SF	CIN	FLA	-MIL	-COL	HOU	SF	-LAD	-SF
COL	MIL	-SD	-NYM	ATL	WAS	-FLA	-ARZ	SF	SD	LAD	-PHI	-WAS	CHC
BOS	-NYY	KC	MIN	-TB	-TEX	-BAL	TOR	BAL	-LAA	-NYY	-TOR	DET	NYY
NYY	BOS	TB	-LAA	-TEX	OAK	LAA	BAL	-CWS	-BAL	BOS	DET	-MIN	-BOS
BAL	TB	-TOR	-TB	OAK	SEA	BOS	-NYY	-BOS	NYY	MIN	-SEA	-CLE	-KC
TB	-BAL	-NYY	BAL	BOS	CWS	-TOR	-OAK	-KC	SEA	OAK	LAA	-SEA	-CLE
TOR	TEX	BAL	-CWS	-LAA	-KC	TB	-BOS	-OAK	CLE	CWS	BOS	-TEX	-MIN
CLE	CWS	DET	-TEX	-CWS	MIN	OAK	LAA	-MIN	-TOR	-DET	KC	BAL	TB
DET	KC	-CLE	-KC	SEA	LAA	TEX	-MIN	-LAA	MIN	CLE	-NYY	-BOS	-CWS
CWS	-CLE	-MIN	TOR	CLE	-TB	-SEA	TEX	NYY	-KC	-TOR	MIN	KC	DET
MIN	LAA	CWS	-BOS	-KC	-CLE	KC	DET	CLE	-DET	-BAL	-CWS	NYY	TOR
KC	-DET	-BOS	DET	MIN	TOR	-MIN	-SEA	TB	CWS	TEX	-CLE	-CWS	BAL
OAK	-SEA	LAA	SEA	-BAL	-NYY	-CLE	TB	TOR	-TEX	-TB	TEX	LAA	-SEA
TEX	-TOR	-SEA	CLE	NYY	BOS	-DET	-CWS	SEA	OAK	-KC	-OAK	TOR	-LAA
LAA	-MIN	-OAK	NYY	TOR	-DET	-NYY	-CLE	DET	BOS	SEA	-TB	-OAK	TEX
SEA	OAK	TEX	-OAK	-DET	-BAL	CWS	KC	-TEX	-TB	-LAA	BAL	TB	OAK

Table A40 - Actual MLB Schedule (- indicates home game) – Part I

TEAM	W14	W15	W16	W17	W18	W19	W20	W21	W22	W23	W24	W25	W26
WAS	-NYM	-BAL	SF	SD	HOU	-CIN	-PIT	CLE	DET	-CWS	-KC	BAL	ATL
NYM	WAS	-NYY	-PHI	MIL	SD	-FLA	-SD	BAL	CLE	NYY	-DET	-MIN	FLA
PHI	-CHC	-BOS	NYM	FLA	ATL	-SD	-FLA	BOS	NYY	-MIN	-CLE	TOR	CIN
ATL	-CIN	PIT	FLA	-PIT	-PHI	LAD	ARZ	MIN	-TB	-KC	CWS	-DET	-WAS
FLA	STL	CWS	-ATL	-PHI	-MIL	NYM	PHI	TB	-TEX	-TB	BAL	-SD	-NYM
PIT	-MIL	-ATL	CIN	ATL	-CHC	-SF	WAS	DET	-CWS	-CLE	TEX	OAK	CHC
CHC	PHI	TEX	-LAD	-STL	PIT	HOU	MIL	-CWS	-OAK	-LAA	SEA	CWS	-PIT
CIN	ATL	CLE	-PIT	-HOU	STL	WAS	-SF	-KC	-LAD	SEA	OAK	-CLE	-PHI
STL	-FLA	-LAA	SD	CHC	-CIN	-MIL	LAD	ARZ	-SEA	-OAK	TOR	KC	-ARZ
HOU	-COL	-TB	MIL	CIN	-WAS	-CHC	COL	NYY	KC	-TEX	-SF	TEX	MIL
MIL	PIT	MIN	-HOU	-NYM	FLA	STL	-CHC	-TEX	LAA	COL	-MIN	-SEA	-HOU
ARZ	-SF	-TOR	COL	SF	LAD	-COL	-ATL	-STL	BOS	DET	-NYY	TB	STL
LAD	-SD	-DET	CHC	COL	-ARZ	-ATL	-STL	-LAA	CIN	BOS	LAA	-NYY	SF
SF	ARZ	OAK	-WAS	-ARZ	-COL	PIT	CIN	-OAK	-BAL	TOR	HOU	-BOS	-LAD
SD	LAD	SEA	-STL	-WAS	-NYM	PHI	NYM	-SEA	-TOR	-BAL	TB	FLA	-COL
COL	HOU	KC	-ARZ	-LAD	SF	ARZ	-HOU	-TOR	MIN	-MIL	-BOS	LAA	SD
BOS	-MIN	PHI	TB	-KC	-OAK	BAL	CLE	-PHI	-ARZ	-LAD	COL	SF	-TB
NYY	-TB	NYM	MIN	-CLE	-BAL	TOR	BAL	-HOU	-PHI	-NYM	ARZ	LAD	-SEA
BAL	TEX	WAS	-OAK	TOR	NYY	-BOS	-NYY	-NYM	SF	SD	-FLA	-WAS	-OAK
TB	NYY	HOU	-BOS	-CWS	TOR	TEX	-TOR	-FLA	ATL	FLA	-SD	-ARZ	BOS
TOR	SEA	ARZ	LAA	-BAL	-TB	-NYY	TB	COL	SD	-SF	-STL	-PHI	CLE
CLE	-KC	-CIN	-CWS	NYY	DET	CWS	-BOS	-WAS	-NYM	PIT	PHI	CIN	-TOR
DET	OAK	LAD	SEA	-OAK	-CLE	KC	CWS	-PIT	-WAS	-ARZ	NYM	ATL	MIN
CWS	-LAA	-FLA	CLE	TB	-TEX	-CLE	-DET	CHC	PIT	WAS	-ATL	-CHC	KC
MIN	BOS	-MIL	-NYY	-TEX	SEA	OAK	-KC	-ATL	-COL	PHI	MIL	NYM	-DET
KC	CLE	-COL	-TEX	BOS	-LAA	-DET	MIN	CIN	-HOU	ATL	WAS	-STL	-CWS
OAK	-DET	-SF	BAL	DET	BOS	-MIN	-LAA	SF	CHC	STL	-CIN	-PIT	BAL
TEX	-BAL	-CHC	KC	MIN	CWS	-TB	-SEA	MIL	FLA	HOU	-PIT	-HOU	LAA
LAA	CWS	STL	-TOR	-SEA	KC	SEA	OAK	LAD	-MIL	CHC	-LAD	-COL	-TEX
SEA	-TOR	-SD	-DET	LAA	-MIN	-LAA	TEX	SD	STL	-CIN	-CHC	MIL	NYY

Table A41 - Actual MLB Schedule (- indicates home game) – Part II

TEAM	W27	W28	W29	W30	W31	W32	W33	W34	W35	W36	W37	W38	W39
WAS	-NYM	-SD	-SF	FLA	CIN	MIL	-ATL	-PHI	ARZ	LAD	-FLA	-ARZ	ATL
NYM	WAS	-CIN	-ATL	SF	ARZ	LAD	-STL	-ARZ	ATL	PHI	-COL	-PHI	HOU
PHI	PIT	-ATL	-CIN	CHC	STL	-COL	-ARZ	WAS	FLA	-NYM	-LAD	NYM	-SF
ATL	-FLA	PHI	NYM	-MIL	-SD	FLA	WAS	CIN	-NYM	-SF	HOU	-LAD	-WAS
FLA	ATL	LAD	ARZ	-WAS	-COL	-ATL	SF	SD	-PHI	-STL	WAS	CIN	PIT
PIT	-PHI	HOU	MIL	-HOU	-MIL	-SD	COL	STL	-CIN	-COL	SD	HOU	-FLA
CHC	-CIN	ARZ	LAD	-PHI	-HOU	-STL	HOU	COL	-MIL	-CIN	SF	STL	-SD
CIN	CHC	NYM	PHI	-COL	-WAS	HOU	MIL	-ATL	PIT	CHC	-STL	-FLA	ARZ
STL	-MIL	COL	HOU	-LAD	-PHI	CHC	NYM	-PIT	-HOU	FLA	CIN	-CHC	-MIL
HOU	SD	-PIT	-STL	PIT	CHC	-CIN	-CHC	-MIL	STL	MIL	-ATL	-PIT	-NYM
MIL	STL	-SF	-PIT	ATL	PIT	-WAS	-CIN	HOU	CHC	-HOU	-ARZ	COL	STL
ARZ	-LAD	-CHC	-FLA	SD	-NYM	-SF	PHI	NYM	-WAS	-SD	MIL	WAS	-CIN
LAD	ARZ	-FLA	-CHC	STL	-SF	-NYM	SD	SF	-SD	-WAS	PHI	ATL	-COL
SF	COL	MIL	WAS	-NYM	LAD	ARZ	-FLA	-LAD	COL	ATL	-CHC	-SD	PHI
SD	-HOU	WAS	COL	-ARZ	ATL	PIT	-LAD	-FLA	LAD	ARZ	-PIT	SF	CHC
COL	-SF	-STL	-SD	CIN	FLA	PHI	-PIT	-CHC	-SF	PIT	NYM	-MIL	LAD
BOS	-BAL	TB	TOR	-TEX	OAK	SEA	LAA	-DET	-CLE	NYY	TOR	TEX	-LAA
NYY	-TOR	OAK	SEA	-TB	-LAA	-KC	CLE	TB	-TOR	-BOS	TEX	KC	-DET
BAL	BOS	DET	TEX	-TOR	-TB	-MIN	TOR	KC	-LAA	-CWS	CLE	TB	-SEA
TB	MIN	-BOS	-CLE	NYY	BAL	CLE	-DET	-NYY	-MIN	TOR	DET	-BAL	-TEX
TOR	NYY	-MIN	-BOS	BAL	KC	DET	-BAL	-CLE	NYY	-TB	-BOS	LAA	OAK
CLE	-OAK	TEX	TB	-DET	MIN	-TB	-NYY	TOR	BOS	-MIN	-BAL	-SEA	KC
DET	-SEA	-BAL	-MIN	CLE	-TEX	-TOR	TB	BOS	-CWS	-LAA	-TB	CWS	NYY
CWS	TEX	-LAA	-KC	MIN	SEA	OAK	-SEA	-OAK	DET	BAL	-MIN	-DET	MIN
MIN	-TB	TOR	DET	-CWS	-CLE	BAL	KC	-SEA	TB	CLE	CWS	-OAK	-CWS
KC	LAA	SEA	CWS	-OAK	-TOR	NYY	-MIN	-BAL	OAK	SEA	LAA	-NYY	-CLE
OAK	CLE	-NYY	-LAA	KC	-BOS	-CWS	TEX	CWS	-KC	-TEX	SEA	MIN	-TOR
TEX	-CWS	-CLE	-BAL	BOS	DET	-LAA	-OAK	LAA	SEA	OAK	-NYY	-BOS	TB
LAA	-KC	CWS	OAK	-SEA	NYY	TEX	-BOS	-TEX	BAL	DET	-KC	-TOR	BOS
SEA	DET	-KC	-NYY	LAA	-CWS	-BOS	CWS	MIN	-TEX	-KC	-OAK	CLE	BAL

Table A42 - Actual MLB Schedule (- indicates home game) – Part III

TEAM	W40	W41	W42	W43	W44	W45	W46	W47	W48	W49	W50	W51	W52
WAS	PHI	-CHC	-STL	FLA	PIT	-NYM	-FLA	ATL	PHI	-HOU	-ATL	-PHI	NYM
NYM	PIT	-FLA	-HOU	ATL	CHC	WAS	-PHI	-PIT	-ATL	FLA	PHI	-MIL	-WAS
PHI	-WAS	-HOU	SD	LAD	-MIL	-FLA	NYM	FLA	-WAS	-ATL	-NYM	WAS	ATL
ATL	CHC	COL	-FLA	-NYM	FLA	PIT	-STL	-WAS	NYM	PHI	WAS	-FLA	-PHI
FLA	-HOU	NYM	ATL	-WAS	-ATL	PHI	WAS	-PHI	-CHC	-NYM	MIL	ATL	-PIT
PIT	-NYM	-STL	MIL	CHC	-WAS	-ATL	CIN	NYM	-ARZ	-STL	-HOU	STL	FLA
CHC	-ATL	WAS	CIN	-PIT	-NYM	-HOU	MIL	STL	FLA	-SF	-STL	SD	HOU
CIN	LAD	SF	-CHC	-MIL	STL	COL	-PIT	-ARZ	HOU	MIL	SD	-HOU	-MIL
STL	-SF	PIT	WAS	HOU	-CIN	MIL	ATL	-CHC	-SD	PIT	CHC	-PIT	-COL
HOU	FLA	PHI	NYM	-STL	ARZ	CHC	-LAD	-MIL	-CIN	WAS	PIT	CIN	-CHC
MIL	-SD	-LAD	-PIT	CIN	PHI	-STL	-CHC	HOU	SF	-CIN	-FLA	NYM	CIN
ARZ	-COL	SD	SF	-SD	-HOU	-SF	COL	CIN	PIT	-COL	-LAD	SF	LAD
LAD	-CIN	MIL	COL	-PHI	-SF	SD	HOU	SF	-COL	-SD	ARZ	COL	-ARZ
SF	STL	-CIN	-ARZ	-COL	LAD	ARZ	SD	-LAD	-MIL	CHC	COL	-ARZ	-SD
SD	MIL	-ARZ	-PHI	ARZ	-COL	-LAD	-SF	COL	STL	LAD	-CIN	-CHC	SF
COL	ARZ	-ATL	-LAD	SF	SD	-CIN	-ARZ	-SD	LAD	ARZ	-SF	-LAD	STL
BOS	-TOR	-SEA	TB	BAL	-CWS	-TB	OAK	SEA	-TOR	-BAL	NYY	CWS	-NYY
NYY	-SEA	TOR	CWS	-OAK	-TOR	-BAL	TEX	TB	BAL	-TB	-BOS	TOR	BOS
BAL	-TEX	CWS	LAA	-BOS	-TB	NYY	DET	-TOR	-NYY	BOS	TOR	TB	-DET
TB	OAK	LAA	-BOS	-TOR	BAL	BOS	TOR	-NYY	-LAA	NYY	-SEA	-BAL	KC
TOR	BOS	-NYY	-DET	TB	NYY	-TEX	-TB	BAL	BOS	-SEA	-BAL	-NYY	MIN
CLE	DET	-OAK	-KC	-CWS	SEA	LAA	-MIN	-LAA	KC	MIN	-KC	-DET	CWS
DET	-CLE	-KC	TOR	MIN	KC	-CWS	-BAL	TEX	CWS	-KC	-MIN	CLE	BAL
CWS	KC	-BAL	-NYY	CLE	BOS	DET	-KC	-MIN	-DET	OAK	LAA	-BOS	-CLE
MIN	-LAA	TEX	SEA	-DET	-TEX	-KC	CLE	CWS	-OAK	-CLE	DET	KC	-TOR
KC	-CWS	DET	CLE	-TEX	-DET	MIN	CWS	-OAK	-CLE	DET	CLE	-MIN	-TB
OAK	-TB	CLE	TEX	NYY	-LAA	-SEA	-BOS	KC	MIN	-CWS	-TEX	LAA	SEA
TEX	BAL	-MIN	-OAK	KC	MIN	TOR	-NYY	-DET	SEA	LAA	OAK	-SEA	-LAA
LAA	MIN	-TB	-BAL	SEA	OAK	-CLE	-SEA	CLE	TB	-TEX	-CWS	-OAK	TEX
SEA	NYY	BOS	-MIN	-LAA	-CLE	OAK	LAA	-BOS	-TEX	TOR	TB	TEX	-OAK

Table A43 - Actual MLB Schedule (- indicates home game) – Part IV

BIBLIOGRAPHY

1. Abramson, D. Constructing school timetables using simulatedannealing: sequential and parallel algorithms. Management Science, Volume 37 Issue 1 (1991), pp. 98-113.

2. Abramson, D. A., Dang, H., and Krisnamoorthy, M. Simulated annealing cooling schedules for the school timetabling problem. Asia-Pacific Journal of Operational Research, Volume 16 (1999), pp. 1-22.

3. Abuhamdah ,A. "Experimental result of late acceptance randomized descent algorithm for solving course timetabling problems" IJCSNS International Journal of Computer Science and Network Security Volume 10 No.1 (January 2010).

4. AfikBakur, M. and Aksop C. Integer Programming Approach to a University Timetabling Problem. Hacettepe Journal of Mathematics and Statistics. Volume 37 , No. 1 (2008), 41 – 55.

5. Al Milli, N. Hybrid Genetic Algorithms with Great Deluge For Course Timetabling.IJCSNS International Journal of Computer Science and Network Security, Volume.10 No.4 (April 2010).

6. Anagnostopoulos A., Michel L., Van Hentenryck P., and Vergados Y. A simulated annealing approach to the traveling tournament problem. Journal of Scheduling 2006 Volume 9: pp. 177–93.

7. Araújo, A., Boeres, M.C., Rebello, V.E., Ribeiro, C.C., and Urrutia, S.., Exploring Grid Implementations of Parallel Cooperative Metaheuristics, A Case Study for the Mirrored Traveling Tournament Problem. Chapter 16, Metaheuristics Volume 39, US:Springer, 2007.

8. Armstrong J and Willis RJ . Scheduling in the Cricket World Cup: A case study. 1993. Journal Opl Res Soc 44: pp. 1067–1072.

9. Bar-Noy, A. and Moody, D. “A Tiling Approach for Fast Implementation of the Traveling Tournament Problem,” Practice and Theory of Automated Timetabling (PATAT06, Brno, August 2006), Conference Proceedings, pp. 351-358.

10. Bartsch, T., Drexl, A., Kroger, S. Scheduling the professional soccer leagues of Austria and Germany. Computers and Operations Research 33, 1907–1937 (2006)

11. Bean, J.C. and Birge, J.R. Reducing travelling costs and player fatigue in the National Basketball Association, Interfaces 10 (3) (1980) pp. 98–102.

12. Benoist, T., F. Laburthe, and B. Rottembourg, Lagrange relaxation and constraint programming collaborative schemes for travelling tournament problems. CP-AI-OR’2001, Wye College (Imperial College), Ashford, Kent UK (April 2001).

13. Bhattacharyya, R. A Note on Complexity of Traveling Tournament Problem 2009. Cycle, Issue (2001). pp. 1-7.

14. Briskorn, D. and Drexl A., “A Branching Scheme for Finding Cost-Minimal Round Robin Tournaments", European Journal of Operational Research, Volume 197, No.1 (2009), pp. 68-76.

15. Burke, E.K. and Bykov, Y. “Solving Exam Timetabling Problems with the Flex-Deluge Algorithm,” Practice and and Theory of Automated Timetabling (PATAT06, Brno, August 2006), Conference Proceedings, pp. 351-358.

16. Burke, E.K. and Bykov Y., A Late Acceptance Strategy in Hill-Climbing for Exam Timetabling Problems. P proccedings of PATAT 2008 conference. (PATAT08, Montreal, August 2008), Conference Proceedings..

17. Burke, E.K. and Newall, J.P. A Multi-Stage Evolutionary Algorithm for the Timetable Problem. IEEE Transactions on Evolutionary Computation, Volume. 3.1. pp 63-74.

18. Burke E.K, Marecek, J.,Parkes, A.and Rudová, H., Penalising Patterns in Timetables: Novel Integer Programming Formulation Operations Research Proceedings 2007, series Operations Research Proceedings, volume 2007, pages 409-414. ISSN 0721-5924. Springer, 2008.

19. Burke, E.K., Qu, R., and Soghier A. Adaptive Selection of Heuristics within a GRASP for Exam**.** Timetabling Problems**.** MISTA - 4th Multidisciplinary International Scheduling Conference, Dublin, Ireland, August 2009

20. Bykov, Y. The Description of the Algorithm for the International Timetabling Competition," International Timetabling Competition Results. 31. March 2003. http://www.idsia.ch/Files/ttcomp2002/bykov.pdf.

21. Cain Jr. W.O. A computer-assisted heuristic approach used to schedule the major league baseball clubs. Optimal Strategies in Sports. North-Holland PublishingCompany: New York, USA, 1977. pp 32–41.

22. Cardemil, A. Optimizacion de _xtures deportivos: Estado del arte y un algoritmo tabu searchpara el traveling tournament problem (2002). Master's thesis, Departamento de ComputacionFacultad de Ciencias Exactus y Naturales, Universidad de Buenes Aires, 2002.

23. Christiansen, H. Constraint Solving and Language Processing: First International Workshop, CSLP 2004, Roskilde, Denmark, September 1-3.

24. Connelly, D.T.. General Purpose Simulated Annealing. European Journal of Operations Research, 43, 1992.

25. Cooper T. and Kingston J. The Complexity of Timetable Construction Problems. University of Sydney, Technical Report Number 495. February 1995.

26. Cormen, Thomas H. , Leiserson, Charles E., Rivest Ronald L., and Stein, Clifford. *Introduction to Algorithms*.Boston: McGraw-Hill, 2001. pp. 570-573.

27. Costa, "An evolutionary tabu search algorithm and the NHL scheduling problem", INFOR 33 (3), 1995, pp. 161–178.

28. Courdeau, J.F., Jaumard, Bridgette and Morales, Rodrigo. Efficient Timetabling Solution with Tabu Search. International Timetabling Competition Results, Volume 31. http://www.idsia.ch/Files/ttcomp2002/jaumard.pdf. March 2003

29. Croce, F.D. and Oliveri, D.. Scheduling the Italian football league: an ILP-based approach.Computers and Operations Research, 33(7):1963 (1974).

30. Daskalaki, A., Birbas,T. and Housos, B. An integer programming formulation for a case study in university timetabling.European Journal of Operational Research 153 (2004) pp. 117–135.

31. Davenport A. and Tsang E. Solving Constraint Satisfaction Sequencing Problems by Iterative Repair. Procceedings of the First International Conference on the Practical Applications of Constraint Technologies and Logic Programming (PACLP-99), London (April 1999). pp. 345–357.

32. de Werra, D. An introduction to timetabling. European Journal of Operational Research 19 (1985) . pp. 151-162.

33. Decter, R. *Constraint Processing*, San Francisco: Morgan Kaufmann, 2003. pp. 117-144.

34. Di Gaspero L, and Schaerf A. A composite-solution tabu search approach to the traveling tournament problem. Journal of Heuristics 2007:13 pp. 189–207.

35. Di Gaspero, L. and Schaerf, A. A Multineighbourhood Local Search Solver for the Timetabling Competition TTCOMP 2002, Practice and and Theory of Automated Timetabling (PATAT04: Pittsburgh, August 2004), Conference Proceedings, pp. 475-478.

36. Di Gaspero, L. and Schaerf, A.: Tabu search techniques for examination timetabling. PATAT 2000. LNCS, vol. 2079, Springer,Heidelberg (2001). pp. 104–117.

37. Dinitz, J.H., and Stinson, D.R.. A Hill-climbing Algorithm for the Construction of One-Factorizations and Room Squares. SIAM J. Alg. Disc. Meth., 8(3):430–438, July 1987.

38. Dinitz J.H., E.Lamken, and Wallis, W.D. Scheduling a tournament. Handbook of Combinatorial Designs, CRC Press, 1995. pages 578-584.

39. DiNunzio M, and Kruk S. “Soccer Tournament Scheduling Using Constraint Programming” , Practice and Theory of Automated Timetabling (PATAT10, Belfast, August 2010), Conference Proceedings. pp. 193-200.

40. Dueck G. New optimisation heuristics: The great deluge algorithm and record-to-record travel. Journal of Computational Physics, Volume 104 (1993) pp. 86-92.

41. N10. Durán G., Guajardo M., Miranda J., Sauré D., Souyris S., Weintraub A. Scheduling the Chilean soccer league by integer programming. Interfaces 2007;37 pp. 539–552.

42. Easton, K., Nemhauser G.L. and Trick M.A. The travelling tournament problem: description and benchmarks. In: Walsh, T., editor. Principles and practice of constraint programming. Lecture notes in computer science, vol. 2239. Berlin: Springer; 2001. pp. 580–585.

43. Festa, P. and Resende, M. G. C. An annotated bibliography of GRASP, European Journal of Operational Research (2004). pp. 1 – 50.

44. Fleurent C. and Ferland, J.A. Allocating games for the NHL using integer programming. Operations Research 1993; 41: pp. 649–654.

45. Fujiwara, N., Imahori, S., Matsui T., Miyashiro R. Constructive algorithms for the constant distance traveling tournament problem. Practice and theory of automated timetabling VI. Lecture notes in computer science, vol. 3867. Berlin: Springer; 2007. p. 135–146.

46. Glover, F. and Laguna, M. *Tabu search*, Boston : Kluwer Academic Publishers. 1997.

47. Glover, F. (1986) Future paths for integer programming and links to artificial intelligence, Computers and Operations Research 13, pp. 533-549.

48. Goossens D and Spieksma F. Scheduling the Belgian soccer league. Interfaces 2009; 39: pp. 109–118.

49. Guedes A., Ribeiro C.C. A hybrid heuristic for minimizing weighted carry-over effects in round robin tournaments. Proceedings of the 4th multidisciplinary international conference on scheduling theory and applications, Dublin, 2009.

50. Hadjidj D. and Drias, H. GRASP and Guided Local Search for the Examination Timetabling Problem**.** International Journal of Artificial Intelligence and Soft Computing 2010 - Volume 2, No.1/2 pp. 103 – 114.

51. Hamiez, J and Hao, J. Hao. Solving the sports league scheduling problem with tabu search.In A. Nareyek, editor, Local Search for Planning and Scheduling, volume 2148 of LectureNotes in Computer Science (2001), pages 24-36.

52. Henz,M. Constraint-based round robin tournament planning. In D. De Schreye, editor, Proceedings of the International Conference on Logic Programming, Las Cruces, New Mexico, MIT Press (1999). pages 545-557.

53. Henz, M.,MÄuller, T., and Thiel, S.. Global constraints for round robin tournament scheduling. European Journal of Operational Research, 153 (2004) pp. 92-101.

54. Henz M., Playing with constraint programming and large neighborhood search for traveling tournaments. Practice and Theory of Automated Timetabling (PATAT04, Pittsburgh, August 2004), Conference Proceedings. pp. 23–32.

55. Hertz A. Tabu search for large scale timetabling problems. European Journal of Operations. Research, 54:39–47, 1991.

56. Fen, H.S., Safaai-Deris, I., and Hashim, S. Proceedings of the International MultiConference of Engineers and Computer Scientists 2009 Vol I-IMECS 2009, March 18 - 20, 2009, Hong Kong.

57. Kendall G. Scheduling English football fixtures over holiday periods. Journal of the Operational Research Society 2008; 59:743–55.

58. Kendall, G. and Hussin, N.M.: Tabu Search Hyper-Heuristic Approach to the Examination Timetabling Problem at University Technology MARA. PATAT 2004. LNCS, vol. 3616, pp. 199-218.

59. Kendall G., Knust S., Ribeiro C. C. and Urrutia S. Scheduling in Sports: An Annotated Bibliography. Computers & Operations Research (2009), 37: pp.1-19.

60. Kingston, J. “Hierarchical Timetable Construction”, Practice and Theory of Automated Timetabling (PATAT06: Brno, August 2006), Conference Proceedings, pp. 196-208.

61. Kingston, J. A tiling algorithm for high school timetabling, Proceedings Practice and Theory of Automated Timetabling (PATAT04, Pittsburgh, August 2004). Conference Proceedings, pp. 208-225.

62. .Kingston, J. Timetable Construction: The Algorithms and Complexity Perspective (Plenary Talk) , Practice and Theory of Automated Timetabling (PATAT10, Belfast, August 2010), Conference Proceedings. p. 26-36.

63. Kirkpatrick, S., Gelatt, C. D. and Vecchi, M. P. (1983). Optimisation by simulated annealing, Science, 220, pp. 671-680.

64. Knust, S. and Lucking, D. Minimizing costs in round robin tournaments with place constraints. Computers & Operations Research 2009;36: pp. 2937–2943.

65. Kostuch, P. The University Course Timetabling Problem with a 3-Phase Approach, Practice and and Theory of Automated Timetabling (PATAT04: Pittsburgh, August 2004), Conference Proceedings, pp. 251-266.

66. Langford, G An Improved Neighbourhood for the Traveling Tournament Problem, 1-12. . Retrieved from http://arxiv.org/abs/1007.0501. March 2011.

67. Leong, G.T.: Constraint Programming for the Traveling Tournament Problem., http://www.comp.nus.edu.sg/~henz/. Project Thesis, National University of Singapore (2003).

68. Lewis, R. A general-purpose hill-climbing method for order independent minimum grouping problems:A case study in graph colouring and bin packing. Computers and Operations Research, 2009.

69. Lim, A. and Fu Zhaohui, F., Heuristics for the exam scheduling problem, Tools with Artificial Intelligence, IEEE International Conference, 2009. Proceedings pp. 172-173.

70. Lim,,A. Rodrigues B, and Zhang X. A simulated annealing and hill-climbing algorithm for the traveling tournament problem. European Journnal of Operational Research Febraury 2005.

71. Lim, A. and Zhang, X.Integer programming and simulated annealing for scheduling sports competition on multiple venues, Proceedings of the Fifth Metaheuristics International Conference (MIC 2003).

72. Little League Home website. Little League history. Online documentation. at: http://www.littleleague.org/Learn_More/about/historyandmission/aroundtheworld .htm, March 2011.

73. Magaletta, F. Catholic Youth Organization of Westchester-Putnam.. Online document at http://www.wpcyo.org. March 2011.

74. Mansour N., and Timany M. Stochastic Search Algorithms for Exam Scheduling, International Journal of Computational Intelligence Research, Vol.3, No.4 (2007), pp. 353–361.

75. Mauritsius T., Berretta, R., and Mendes, A.. A hybrid simulated annealing with Kempechain neighborhood for the university timetabling problem. In Proc. 6th ACIS Int. Conf. on Computerand Information Science (ACIS-ICIS), 2007. pages 400–405.

76. McAloon, K. , Tretko, C., and Wetzel, G.. Sports league scheduling. In Proceeding of the 3rdILOG Optimization Suite International Users Conference, Paris, 1997.

77. McCollum, B. University Timetabling: Bridging the Gap between Research and Practice, Practice and Theory of Automated Timetabling (PATAT06: Brno, August 2006), Conference Proceedings, pp. 15-33.

78. McCollum, B., McMullan, P. J., Parkes, A. J; Burke, E. K. and Abdullah, S. An Extended Great Deluge Approach to the Examination Timetabling Problem. Proceedings of the 4th Multidisciplinary International Scheduling Conference: Theory and Applications (MISTA 2009), 10-12 Aug 2009, Dublin, Ireland, pp. 424-434.

79. McMullan, P.. An Extended Implementation of the Great Deluge Algorithm for Course Timetabling, Lecture Notes in Computer Science, Springer, Vol 4487 (2007), pp. 538-545.

80. Melício, F., P. Caldeira, and A. Rosa. Solving the Timetabling Problem with Simulated Annealing. Proc. First Int. Conf. on Enterprise Information Systems, ICEIS' 99 (199) pp. 272-279.

81. Moody, D, Kendall, G. and Bar-Noy, A.. “An efficient and robust approach to generate high quality solutions for the Traveling Tournament Problem” , Practice and Theory of Automated Timetabling (PATAT10, Belfast, August 2010), Conference Proceedings. pp. 273-282.

82. Moura A. and Scaraficci R. A GRASP Strategy for a More Constrained School Timetabling Problem. International Journal of Operational Research, Vol. 7, No. 2. (2010). pp. 152-170.

83. Muhlenthaler, M. and Wanka R. A Novel Insertion Strategy for Creating Feasible Course Timetables , Practice and Theory of Automated Timetabling (PATAT10, Belfast, August 2010), Conference Proceedings. pp. 294-304.

84. Noronha T.F., Ribeiro C.C., Duran G., Souyris S., and Weintraub A. A branch-and-cut algorithm for scheduling the highly-constrained Chilean soccer tournament. Practice and theory of automated timetabling VI. Lecture notes in computer science, vol. 3867. Berlin: Springer; 2007. pp. 174–186.

85. Nandhini, M . and Kanmani, S. A Survey of Simulated Annealing Methodology for University Course Timetabling. Information Paper.International Journal of Recent Trends in Engineering, Volume. 1, No. 2, May 2009.

86. Nnuohiro,E. and Maklin,K. Applying Multi-Agent Algorithm to a Class Scheduling System,,Journal of Advanced Computational Intelligence and Intelligent Informatics, Vol.9 No.3 (May 2005), pp. 314-320.

87. O’Brien, J. Brooklyn Christian Basketball Association. Online document at http://www.bcba.com. March 2011.

88. Ozcan**,** E. Final Exam Scheduler **(**FES**),** Proc. of 2005 IEEE Congress on Evolutionary Computation, Volume 2. 2005

89. Ozcan E., Bykov Y., Birben M., and Burke, E. K. Examination timetabling using late acceptance hyper-heuristics, in proccedings of the 11th conference on Congress of Evolutionary Computation. Trondheim, Norway, May 2009

90. Paechter, B. International Timetabling Competition. Metaheuristics Network. 31 March 2003.

91. Post G. and Woeginger G. Sports tournaments, home–away assignments, and the break minimization problem. Discrete Optimization Volume 3. (2006) pp. 165–73.

92. Qu R., Burke E.K., McCollum, B., Merlot, L.T.G., and Lee, S.Y. A survey of search methodologies and automated approaches for examination timetabling, Technical Report NOTT-CSTR- 2006-04, School of Computer Science & IT, University of Nottingham.

93. PATAT2008 International Timetabling Competition. Online document at http://www.cs.qub.ac.uk/itc2007. Retrieved: November 29, 2007.

94. Rasmussen R.V., and Trick M.A.. Round robin scheduling—a survey. European Journal of Operational Research 2008;Volume 188. pp. 617–636.

95. Rasmussen R.V. Scheduling a triple round robin tournament for the best Danish soccer league. European Journal of Operational Research 2008 Volume 185: pp. 95–810.

96. Rasmussen, R.V and Trick MA. A Benders approach for the constrained minimum break problem. European Journal of Operational Research 2007 Volume 177. pp. 198–213.

97. Regin J.C. Minimization of the number of breaks in sports scheduling problems using constraint programming. DIMACS series in discrete mathematics and theoretical computer science, Volume 57 (2001). pp. 115–130.

98. Ribeiro C.C., and Urrutia S. Heuristics for the mirrored traveling tournament problem. European Journal of Operational Research 2007. Volume 179. pp. 775–87.

99. Ribeiro C.C., Urrutia S. Scheduling the Brazilian soccer tournament with fairness and broadcast objectives. Practice and theory of automated timetabling VI. Lecture notes in computer science, volume 3867. Berlin: Springer; 2007. pp. 147–157.

100. Ribeiro, C.C. and Urrutia, S, Scheduling the Brazilian soccer tournament by integer programming maximising audience shares under fairness constraints. Proceedings of the 2nd International Conference on the Mathematics in Sport, Groningen, Netherlands, 2009.

101. Russell, R.A. and Urban, T.L.. A constraint programming approach to the multiple-venue,sport-scheduling problem. Computers & Operations Research, volume 33 (2006) pp. 1895-1906.

102. Schaerf, A., Tabu Search Techniques for Large High-school Timetabling Problems , Proceedings of the Thirteenth National Conference on Artificial Intelligence AAAI Press. (1996). pp. 363-368.

103. Schaerf A. Scheduling sport tournaments using constraint logic programming. Constraints 1999;4. pp. 43–65.

104. Schreuder, J. A. M.: Combinatorial aspects of construction of competition Dutch professional football leagues. Discrete Applied Mathematics, Volume. 35(3). (1992). pp.301-312.

105. Shen, H., and Zhang, H. (2004) *Greedy Big Steps as a Meta-Heuristic for Greedy Big Steps.* http://goedel.cs.uiowa.edu/AR-group/readings/aaai_ttp.pdf. The University of Iowa AR Reading Group, Spring 2004.

106. Souza M, Ochi L, Maculan N A grasp-tabu search algorithm for solving school timetabling problems. Metaheuristics: Computer Decision-Making, Kluwer Academic Publishers, Boston, (2003) pp 659–672 .

107. Thompson, J. M. and Dowsland, K. A . General cooling schedules for a simulated annealing based timetabling system. In: Practice and Theory of Automated Timetabling, Lecture Notes in Computer Science, 1153, Springer-Verlag, (1996). pp. 345-363.

108. Thompson, J. M. and Dowsland, K. A. A robust simulated annealing based examination timetabling system. Computers Operational Research Volume 25, No 7/8, (1998). pp. 637-648.

109. Trick, M.A., Challenge Traveling tournament instances. Online document at http://mat.gsia.cmu.edu/TOURN/. March 2011

110. Trick, M.A., Traveling Umpire problem. Online document at http://mat.gsia.cmu.edu/TUP/. March 2011

111. Trick, M.A., TTP Relaxed constraints and bye-weeks. Online document at http://mat.gsia.cmu.edu/TOURN/relaxed. March 2011

112. Trick M.A. Formulations and Reformulations in Integer Programming. Lecture Notes In Computer Science; Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, Springer 2005 Volume 3524/2005 pp: 366-379..

113. Trick, M.A. Sports Scheduling and Advances in Integer and Constraint Programming, New York University, New York, New York, April 15,2010.

114. Trick, M.A.. and Bao, R. The Relaxed Traveling Tournament Problem , Practice and Theory of Automated Timetabling (PATAT10, Belfast, August 2010), Conference Proceedings. pp. 472-476.

115. Van Hentenryck, P. and Vergados Y. Population-Based Simulated Annealing for Traveling Tournaments, AAAI - National Conference on Artificial Intelligence, 2007.

116. Wang X., Huang L. , and Zou, Qing, The Research on Multi-Strategy Course Scheduling Algorithm. Information Science and Engineering, International Conference (2009), pp. 2419-2422.

117. White, G. M. and Xie, B. S. Examination timetables and tabu search with longer term memory. In: Practice and Theory of Automated Timetabling III , Lecture Notes in Computer Science, Springer-Verlag, (2001). pp. 85-103.

118. White, G.M., Xie, B.S. and Zonjic, S. (2004) Using tabu search with longer-term memory and relaxation to create examination timetables.European Journal of Operational Research 153, pp. 80–91.

119. Willis RJ, Terrill BJ. Scheduling the Australian state cricket season using simulated annealing. Journal of the Operational Re- search Society (1994). pp. 276–80.

120. Wright. M.B. Scheduling Fixtures for basketball New Zealand. Computers & Operations Research, Volume 33 (2006). pp. 1875-1893.

121. Wright M. Scheduling fixtures for New Zealand cricket. IMA Journal of Management Mathematics Volume 16 (2005) pp. 99–112.