A Computational Study of Problems in Sports

by

Tyrel Clinton Russell

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Computer Science

Waterloo, Ontario, Canada, 2010

Abstract

This thesis examines three computational problems in sports. The first problem addressed is determining the minimum number of points needed to guarantee qualification for the playoffs and the minimum number of points needed to have a possibility of qualification for the playoffs of the National Hockey League (NHL). The problem is solved using a phased approach that incrementally adds more complicated tie-breaking constraints if a solution is not found. Each of the phases is solved using a combination of network flows, enumeration and constraint programming. The experimental results show that the solver efficiently solves instances at any point of the season. The second problem addressed is determining the complexity, either worst-case theoretical or practical, of manipulation strategies in sports tournaments. The two most common types of competitions, cups and round robins, are considered and it is shown that there exists a number of polynomial time algorithms for finding manipulation strategies in basic cups and round robins as well as variants. A different type of manipulation, seeding manipulation, is examined from a practical perspective. While the theoretical worst-case complexity remains open, this work shows that, at least on random instances, seeding manipulation even with additional restrictions remains practically manipulable. The third problem addressed is determining whether manipulation strategies can be detected if they were executed in a real tournament. For cups and round robins, algorithms are presented which identify whether a coalition is manipulating the tournament with high accuracy. For seeding manipulation, it is determined that even with many different restrictions it is difficult to determine if manipulation has occurred.

Acknowledgements

I have been helped by a number of people in the completion of this work and I would like to take the opportunity to thank those people for their support during my time at the University of Waterloo. I would like to thank my supervisor Peter van Beek for his dedication, advice, counsel and ideas during the completion of this work. I would also like to thank Toby Walsh for his advice and for his input and ideas on the publication that we co-authored. I would like to thank my external examiner Michael Trick for his input and his corrections which helped make my thesis better. As well, the other members of my committee, Alex Lopez-Ortiz, Dan Brown and Ada Barlatt, for their corrections and insightful comments on the thesis. I would like to thank Hagit for her time, care and love without which I would not have been able to succeed. I would like to thank my former roommates, Jeff Quilliam, Laurent Charlin and Fabienne Beduneau, for their laughter and friendship. I would like to thank my friends for helping me stay sane and relaxed. I would also like to thank the members of the AI Lab, past and present, for their ideas and input. Last, but not least, I would like to thank my family for their support and encouragement.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Computational techniques have been applied to sports since at least the mid-sixties when Schwartz [63] developed a flow-based technique to solve when teams had clinched a baseball title. Since then other problems have been tackled using computational techniques. Examples include tournament scheduling [49], league planning [62], ranking [44] and elimination problems [63]. Additionally, in the last couple of decades, there has been a large body of work examining the computational properties of election manipulation. These type of manipulations can be applied to sports as both elections and sports competitions fall into the broader category of social choice mechanisms.

The area of determining when teams have clinched a sporting title has been expanded into a host of other papers including both practical results and complexity studies. The first problem addressed in this thesis is qualification and elimination in hockey, one of the sports that has received very little attention from computational studies. The problem differs from other qualification and elimination problems in other sports [1, 54] in that the scoring mechanism is different than those traditionally used and the method of qualification combines the idea of wild cards and multiple playoff positions.

Aside from its intrinsic interest as a difficult combinatorial problem, the problem has a practical interest to the fans that watch the sport. In Canada, major national daily newspapers report whether the teams have clinched or been eliminated from the playoffs. The methods used by the papers are heuristic and the results can vary from the optimal reporting date by several days. Beyond knowing the date of elimination, it is possible to calculate exactly the number of games needed to clinch a playoff spot, to guarantee for certain that the team will qualify, and the number of games the team could lose and still earn a playoff spot. From a management perspective, knowing the exact number of games that must be won to control their own destiny could be helpful in planning tactics and training.

Some of the core attributes of sports are supposed to be integrity, honesty and sportsmanship. However, there have been numerous cases where the desire to win at all cost has pushed coaches, teams and individual athletes to rig games, accept bribes and take performance enhancing drugs. Given that such manipulations happen, it becomes interesting to determine how easily manipulations could be used to change the result of a competition. Research into computational social choice has long studied how coalitions of manipulating agents could change the result of an election given enough information. The subject of manipulating in sports has also been mentioned but with less focus, perhaps due to the fact that elections are used to select the leaders of countries as well as possibly judges and law enforcement officers. However, given the wide impact of sports on society, it is interesting to determine how the techniques from elections can be applied to sports.

The study of manipulation has been focused on computational complexity results. The reason for this examination is the hope that computational complexity might provide a barrier to manipulation. Given a recent experimental study and observations from other researchers, it seems likely that this is not the case [68, 10]. The overall result is that computationally easy problems are easy to manipulate and computationally complex, NP-Hard or stronger, may be hard to manipulate in worst-case but are often easy in practice. However, as shown later, the common types of sports competitions, used widely in professional and amateur sports, are easily manipulable and thus even the weak barrier of computational complexity does not apply. Since it is unlikely that every sports competition is going to change the format of their competitions, there is a benefit to understanding the mechanisms with which a coalition could manipulate the tournament so that the organizers can protect against the possible application of such behaviour by the competitors. This situation is analogous to security researchers that attempt to find exploits in programs so that the weakness can be fixed or managed.

This thesis focuses on two different manipulation strategies: manipulation by a coalition of teams or athletes via thrown games and manipulation by the organizer via schedule generation. For manipulations by a coalition of teams, this thesis examines the complexity of the two common types of sports competitions, round robins and cup competitions, to determine if the ease of manipulation in elections translates to easy manipulations in sports. Beyond their existence, it is interesting to determine how hard it is to manipulate a competition optimally. Optimal in this sense refers to the coalition using as little effort as possible to change the result as desired. In the case of seeding manipulation by the organizer, which has unknown complexity, a practical experimentation is made to determine if the problem can be solved regardless of the complexity. This technique has been used previously on problems of unknown complexity such as, for example, graph isomorphism.

While determining how hard it is to achieve a manipulation is interesting and useful information, it does not provide the organizers of the tournaments with tools to combat the manipulations. Paired with the existence of easy manipulations and the widespread use of such competition mechanisms, detection methods for such manipulations is a crucial tool. Somewhat surprisingly, the work of manipulation has not been paired with detection methods in the literature. Given the assumption that a coalition of agents is changing the results of matches to rig the competition, the changes should be visible to the viewers of the game, for example, by the game not finishing as expected. Currently, almost all sports cheating detection is done through the analysis of betting patterns. If a coalition is actively at work in a competition then the end result should be that some of the unexpected results are those of the coalition. The question that is interesting is whether it is possible to look for patterns in the results to determine the existence of and makeup of the hidden coalition that formed at the start of the tournament.

Another interesting question is whether detection is possible. If a random set of upsets always appears to be the same as a set of manipulations, it would not be possible to differentiate the two sets from the pattern of results. For each of the different type of manipulation, the question of whether it is possible to distinguish between a set of outcomes that happen randomly or via manipulation is addressed.

The rest of the thesis is organized as follows. In Chapter 2, the background material needed to understand the material in the following three chapters is presented. In Chapter 3, a hybrid constraint programming and enumeration approach for solving the qualification and elimination problems related to the NHL hockey is presented. A basic model is proposed and then extended allowing the solver to handle practical instances. In Chapter 4, a complexity analysis is applied to the problems of manipulation in cup competitions and round robins. Without a complexity result for seeding manipulation, a known open problem, constraint programming techniques for solving subgraph isomorphism problems are applied to different models of seeding manipulation. In Chapter 5, detection is discussed as a computational task. For cup competitions, a dynamic programming algorithm is proposed for finding manipulations in cup results. The detectability of round robin and seeding manipulations is also examined. Chapter 6 concludes the results and restates some of the problems open for future work.

Chapter 2

Background

The material in this chapter covers a wide variety of techniques and concepts related to the problems and algorithms described in the remainder of the thesis. A brief introduction to each area is given along with more detail descriptions of the relevant techniques which are used in the thesis. Some of the concepts and techniques are used in more than one chapter of the thesis as listed at the beginning of each section.

2.1 Constraint Programming

Constraint programming is a methodology for solving combinatorial problems using search and logical inference that is used in Chapters 3 and 5. A *Constraint Satisfaction Problem* (CSP) is a mathematical model of a problem consisting of variables, the domains of the variables and constraints on the variables. Variables represent some quantitative aspect of a problem and the domain of that variable is the set of possible values which the variable could be assigned, denoted $dom(x)$ where x is a variable. A constraint on a variable or variables is a restriction of the domain values. For a more detailed introduction to constraint programming, refer to the following sources [29, 43, 56].

Example 2.1. *Given 10 pennies, 8 nickels, 5 dimes and 3 quarters, find a subset of the change such that the monetary value is one hundred cents and there are more pennies than nickels, nickels than dimes, and dimes than quarters. To model this problem as a CSP, four variables are introduced that represent the number of pennies (x_p), nickels (x_n), dimes (x_d) and quarters (x_q). Since there may be none of the coins used or all of the coins used, $dom(x_p) = \{0, \ldots, 10\}$, $dom(x_n) = \{0, \ldots, 8\}$, $dom(x_d) = \{0, \ldots, 5\}$ and of $dom(x_q) = \{0, \ldots, 3\}$. The constraints represent the sum of the coin's values ($x_p + 5x_n + 10x_d + 25x_q = 100$), the number of pennies is greater than the number of nickels ($x_p > x_n$),* *the number of nickels is greater than the number of dimes ($x_n > x_d$) and the number of dimes is greater the number of quarters ($x_d > x_q$)*

An *assignment* of a variable is the association of a particular value with a given variable. A *partial assignment* of a set of variables is a particular assignment of values to a subset of the variables. A *complete assignment* of a set of variables is the assignment of values to every variable in the set.

Example 2.2. *Referring to the CSP described in Example 2.1, an assignment is, for example, associating the value* 4 *with the variable* x_n*. A partial assignment of the variables* $\{x_p, x_n, x_d, x_q\}$ *is* $x_n = 4$ *and* $x_d = 3$*. A complete assignment of the variables* $\{x_p, x_n, x_d, x_q\}$ *is* $x_p = 10$*,* $x_n = 5$*,* $x_d = 4$ *and* $x_q = 1$*.*

The *scope* of a constraint is the set of variables affected by the constraint. A constraint is *violated* if given a partial, possibly complete, assignment of values to the variables in the scope of the constraint there exists no solution. A constraint is *satisfied* if every variable has value and the constraint is not violated. A constraint in a CSP is *Arc Consistent* if for every variable-value pair in the scope of the constraint there exists a value for all other variables which satisfies the constraint.

Example 2.3. *Referring to the CSP described in Example 2.1, the scopes of the four constraints are* $\{x_p, x_n, x_d, x_q\}$*,* $\{x_p, x_n\}$*,* $\{x_n, x_d\}$ *and* $\{x_d, x_q\}$*, respectively. Given the constraint* $x_n > x_d$ *and the assignment of* $x_n = 0$*, the constraint would be violated since the partial assignment allows for no value of* x_d*. If however,* $x_n = 1$ *and* $x_d = 0$*, the constraint* $x_n > x_d$ *is satisfied. Again referring to the constraint* $x_n > x_d$ *and with* $dom(x_n) = \{0, \ldots, 8\}$ *and* $dom(x_d) = \{0, \ldots, 5\}$ *to establish arc consistency, it suffices to ensure that for every value in* x_n *there exists a value in the domain of* x_d *which satisfies the constraint. For example, it has already been shown that* $x_n = 0$ *violates the constraint and therefore the domain of* x_n *can be contracted to* $\{1, \ldots, 8\}$*. For all other values there is a valid assignment for the other variable that satisfies the constraint and no more reductions to the domains can be applied.*

The method by which a constraint is made consistent is known as *propagation.* A CSP is *unsatisfiable* if there exists one or more variables such that there is no value which does not violate at least one constraint. A CSP is *satisfiable* if there exists a value for every variable that simultaneously satisfies every constraint.

Example 2.4. *The CSP described in Example 2.1 is satisfiable since the solution* $x_p = 10$*,* $x_n = 5$*,* $x_d = 4$ *and* $x_q = 1$ *satisfies each constraint.*

Finding a satisfiable solution to a problem is the main task of constraint programming. The application of arc consistency prunes some values from the domains of the variables but does not necessarily result in a solution. Search is used to solve these problems, with backtracking search, which maintains arc consistency at each node in the tree, being the most common search applied in constraint programming. A backtracking search is a depth first search which returns to a previous partial solution when a partial solution denoted by the branch is shown to be unsatisfiable. To maintain arc consistency, arc consistency is applied at each node of the tree to prune the inconsistent values from the variables.

To implement a search, two heuristics must be defined: the variable ordering heuristic and the value ordering heuristic. A *variable ordering heuristic* is a heuristic which selects the best variable to assign a value given a partial assignment of other variables. A *value ordering heuristic* is a heuristic which selects the best value given a variable and a partial assignment of the other variables.

Example 2.5. *Referring to the CSP described in Example 2.1, solving this problem can be accomplished using backtracking search. The variable ordering is to select the variable with the least number of values in its domain and the value ordering is to select the largest value. Applying generalized arc consistency before search yields the values seen in the first row of Table 2.1. Since x_p has the fewest number of values, it is selected and the child representing the largest value is expanded as in Figure 2.1. The next variable is selected is x_q and the largest value remaining in the domain is 2. After applying arc consistency, the domain of x_d is wiped out and a fail is generated. Therefore, a backtrack occurs and the next largest value of x_q, 1, is tried as shown in Figure 2.1 and the domains are pruned as in the fourth row of Table 2.1. The last step selects x_n with value 7 and, after propagation, the solution $x_p = 10$, $x_n = 7$, $x_d = 3$ and $x_q = 1$ is found.*

2.2 Enumeration Techniques

While solving combinatorial problems via search and inference can be effective, it can be advantageous to solve problems, especially smaller problems, using an enumeration technique. An *enumeration technique* is a method which systematically lists all of the possible combinations of values possible for a given problem. The advantage of an enumeration technique over a search technique with inference is that it can be simpler and more efficient to verify whether an assignment of values is a solution for an exponential search space than applying the logical inference while searching.

Example 2.6. *Referring to the problem in Example 2.1, it is possible to apply enumeration to this problem. There are four variables with domains of size 11, 9, 6 and 4. The sum constraint can be checked in time linear in the number of variables and the inequality constraints can be checked in constant time. Therefore, each solution can be checked in linear time. Since there are at most 2376 possible combinations to check, this problem could be solved easily using enumeration.*

Table 2.1: The domains of the variables and the partial solutions for each step of applying backtracking search while maintaining arc consistency. Each row represents the domains after arc consistency has been applied. Row 0 represents the initial domains before any propagation has been applied.

	Domains	Partial Assignment
0	$dom(x_p) = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$	$x_p =?$
	$dom(x_n) = \{0, 1, 2, 3, 4, 5, 6, 7, 8\}$	$x_n =?$
	$dom(x_d) = \{0, 1, 2, 3, 4, 5\}$	$x_d =?$
	$dom(x_q) = \{0, 1, 2, 3\}$	$x_q =?$
1	$dom(x_p) = \{5, 10\}$	$x_p =?$
	$dom(x_n) = \{2, 3, 4, 5, 6, 7, 8\}$	$x_n =?$
	$dom(x_d) = \{1, 2, 3, 4, 5\}$	$x_d =?$
	$dom(x_q) = \{0, 1, 2, 3\}$	$x_q =?$
2	$dom(x_p) = \{10\}$	$x_p = 10$
	$dom(x_n) = \{2, 3, 4, 5, 6, 7, 8\}$	$x_n =?$
	$dom(x_d) = \{1, 2, 3, 4, 5\}$	$x_d =?$
	$dom(x_q) = \{0, 1, 2\}$	$x_q =?$
3	$dom(x_p) = \{10\}$	$x_p = 10$
	$dom(x_n) = \{4, 5, 6, 7, 8\}$	$x_n =?$
	$dom(x_d) = \{\}$	$x_d =?$
	$dom(x_q) = \{2\}$	$x_q = 2$
4	$dom(x_p) = \{10\}$	$x_p = 10$
	$dom(x_n) = \{5, 7\}$	$x_n =?$
	$dom(x_d) = \{3, 4\}$	$x_d =?$
	$dom(x_q) = \{1\}$	$x_q = 1$
5	$dom(x_p) = \{10\}$	$x_p = 10$
	$dom(x_n) = \{7\}$	$x_n = 7$
	$dom(x_d) = \{3\}$	$x_d = 3$
	$dom(x_q) = \{1\}$	$x_q = 1$

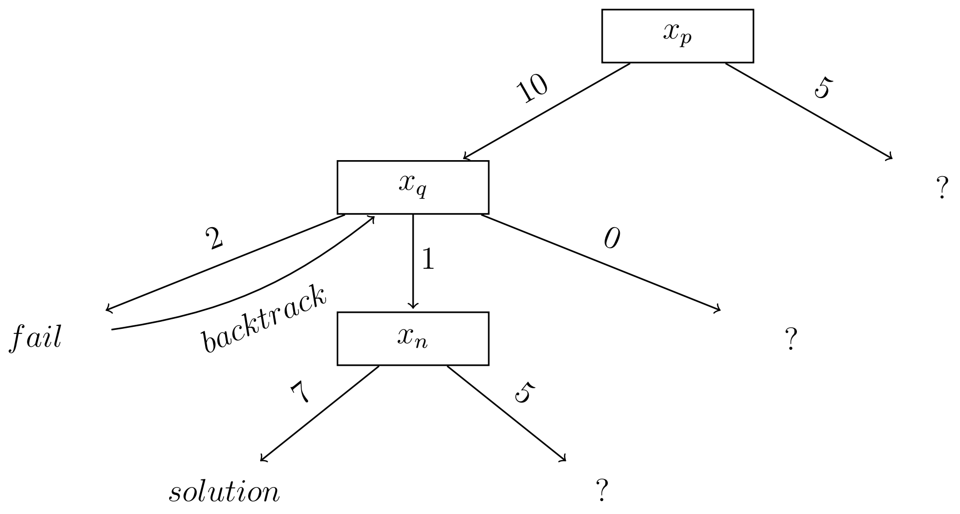


Figure 2.1: The backtracking tree described in Example 2.5. The first step selects 10 from the domain of x_p. Next, x_q is selected with value 2. A domain wipeout occurs and the algorithm backtracks. The next largest value of x_q is 1. Finally, x_n is selected with value 7 and, after propagation, a solution is found.

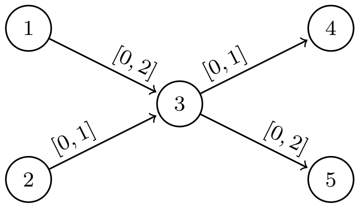
2.3 Network Flows

Network flows are used to solve problems in Chapters 3 and 4. For more information on network flows, refer to the material in [19, 2].

A *flow network* is a directed graph labelled with capacities on each edge, where the capacities must be greater than or equal to zero. A *source node* is a node with no incoming edges. A *sink node* is a node with no outgoing edges. A flow network can always be converted to a single sourced and single sinked flow network by adding two additional nodes and adding edges of infinite capacity from the new source node to the original source nodes and to the new sink node from the original sink nodes (see Figure 2.2)[19]. A *feasible flow* on the graph must satisfy the constraint that the flow is less than the capacity and the sum of the flow entering a node must equal the sum of the flow exiting the node. The value of the flow is equal to the sum of the flow on all of the edges entering the sink node.

The *max flow* for a network is defined as the maximum possible value of the flow and can be determined in polynomial time [19, 2]. A feasible flow is a flow, possibly maximal, which satisfies all the lower and upper bound constraints. The bold values in Figure 2.2b represents a maximum and feasible flow for the network.

If the problem is to find a feasible flow with lower and upper bound capacities, it is



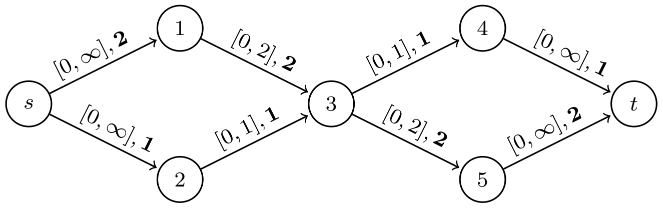


Figure 2.2: (a) The original flow network with five nodes. (b) The new flow network with two additional nodes s and t. s has two outgoing edges to the two original sources 1 and 2 and t has two incoming edges from the two original sinks 4 and 5. The bold numbers represent a maximum and feasible flow.

possible to convert that network into a flow network using only upper bound capacities by adding two additional nodes and extra edges. A max flow is then calculated on the new network. If the value of the max flow is equal to the sum of the capacities entering the new sink node then the problem has a feasible flow which satisfies both the upper and lower bound constraints [2].

2.4 Dynamic Programming

Dynamic programming is a technique for solving problems that relies on optimal substructure within subproblems and overlapping subproblems to solve problems efficiently [19]. Dynamic programming is used in Chapters 4 and 5. For more information, refer to the explanation in [19].

The standard dynamic programming approach is a bottom up approach [19]. As subproblems are solved, the solutions to the subproblems are stored and combined in larger subproblems. This approach works when the result of the subproblem is optimal or is said to have optimal substructure. This means that the optimal solution to the subproblem forms part of the optimal solution to the entire problem.

Keeping track of the subproblems is only beneficial if the problems have overlap. If most subproblems are only solved once then there is no benefit to keeping a table of the solutions to the subproblems.

A top-down approach to solving dynamic programming problems can also be used, termed memoization. Like a recursive algorithm, the sub-problems are solved top down. However, unlike a recursive solution, each solution is stored and if the same subproblem is subsequently encountered then the solution is returned immediately without further recursion. This type of dynamic program can work well in problems with high overlap [19].

2.5 Competitions, Tournaments and Manipulations

In many sporting competitions, the final winner of a competition is decided by a tree-like structure, called a *cup*. The most common type is a *single elimination cup*, a tree structure where the root and internal nodes represent games and leaves represent the teams in the tournament. A *double elimination cup* is two trees where the losers of the primary tree are demoted into the second tree at specified entry points. Losers within the second tree are then eliminated from the tournament. At the end of the competition, the winner of both trees meet and play one or two games. If the winner of the primary tree wins the game then the competitions is finished and that team is declared the winner. If not then, in order to be eliminated twice, they must play a second game which ultimately determines the winner of the competition.

Cups need to be *seeded* to determine which teams play against each other in each round. One method for seeding is by rank. The most common method for ranked seeding is to have the top team play the worst team, the second place team play the second worst team and so forth. An example of ranked seeding using this method is the National Basketball Association in the US. Another method for determining seeding is randomly, also known as a draw. An example of this is the UEFA Champions League where teams reaching the quarter finals are randomly paired for the remainder of the tournament. Seeding may also be more complex (for instance, it may be based on the group from which teams qualify or some other criteria).

Another way that cups are modified is between fixed and unfixed cups. A *fixed cup* is a cup where there is a single seeding at the start of the cup. Examples of this are the National Basketball Association playoffs and the FIFA World Cup of Football. An *unfixed cup* is one where seeding may occur not only before the start but between any round. Examples with an unfixed cup are the National Hockey League playoffs and the UEFA Champions League.

In this thesis, perfectly balanced cup competitions are examined and it is assumed that m, the number of teams, is a power of 2. The names of the teams are t_1, t_2, $\ldots$, t_{2^n}, for some $n = \log m$. A competition tree is a complete binary tree where the leaves are labeled with the set of teams, called the *seeding* of the tournament. The matches in the competition are the games between two teams at each node (except the leaves) of the competition tree. There are exactly $m - 1$ matches in a competition tree. A *round* of a competition tree is defined to be all of the matches occurring at an equal height from the leaves of the tree. The *round* k for $1 \leq k \leq n$ is defined to be all matches at a height k from the leaves.

Finally, a *round robin competition* is a competition where each team plays every other team a given number of times. In a *single round robin competition*, each team plays every other team exactly once. Another common variant of this is for teams to play a *double round robin competition* where each team plays every other team twice, often at home and away.

The set of teams in the competition is denoted T where $|T| = m$. A *tournament* is a directed graph $G = (T, E)$ where the underlying undirected graph is a complete graph. It is assumed that the tournament is known for the remainder of this work. Every directed edge $(t_i, t_j) \in E$ represents a game where t_i is expected to win over t_j. As in election manipulation where the electoral vote is assumed to be known, it is assume that G is known, via an oracle, and, from G, the relative strengths of teams is known and the expected winner of the contests can be determined.

An upset is a match where team i was expected to win over team j according to the tournament graph, but team j won against team i in the actual competition. A team i is said to have *caused* an upset if they lost a match that they were expected to win.

Definition 2.1 (upset)**.** *Let $G = (T, E)$ be a tournament graph. An* upset *is a pair (t_j, t_i) where $(t_i, t_j) \in E$ but t_j won against t_i in some round of the actual competition. Let U denote the set of all upsets that occurred in the competition, let U^k denote only those upsets that occurred at round k, $1 \leq k \leq n$, of the competition, and let U^k_S, $S \subseteq T$, denote only those upsets that occurred at round k that were caused by a team in S; i.e., $U^k_S = \{(t_j, t_i) \mid (t_j, t_i) \in U^k \wedge t_i \in S\}$.*

This thesis focuses on a particular subset of upsets which are caused by a coalition. A *coalition* is a set of teams S, $S \subseteq T$, which conspires to manipulate the competition to change the winner.. A *manipulation* is an upset (t_j, t_i), either executed or planned, which is intentional; i.e., team t_i threw the match or planned to throw the match.

In this thesis, manipulations are restricted to those manipulations where a coalition member loses. Therefore, an edge (t_i, t_j) is manipulable if and only if candidate t_i is a member of the coalition. This restricts the behaviour of the manipulators to throwing games where they could have won. This restriction is due to the fact that it is simple to perform worse but more difficult to play better.

Two different types of manipulation strategies are considered in this work. A *constructive manipulation* is one that ensures a specific team wins the competition. A *destructive manipulation* is one that ensures a specific team loses the competition.

For round robin competitions, the concept of the tournament is generalized beyond the simple win-loss scoring model to a complete graph where the edge $(t_i, t_j) \in E$ has a non-negative weight w_{ij} which represents the number of points that would be earned by t_i when playing t_j in a fair game. A manipulation in this case is defined as an outcome where the points earned in the match are different from those given by the tournament.

However, as before, manipulations are restricted so that the manipulator achieves no more points and the team being manipulated achieves no less points.

The probabilistic variant of the tournament is defined to be a complete directed graph where each edge is labelled with the probability of the outcome. Again, it is assumed that the probabilities are known via some oracle.

2.6 Accuracy, Precision and Recall

When evaluating experimental results, as in Chapter 5, it is sometimes advantageous to look at measures beyond accuracy. In this section, some additional measures are introduced which provide a more thorough picture of data. Information in this section can be found in more detail in [42, sec. 8.1, pg. 267].

In this section, it is assumed there exists some classification task; i.e., a problem of labelling whether a given instance is a member of the class or not. A classification algorithm is a procedure that takes an instance and returns true if an instance is a member of the class and false otherwise. A *true positive* (tp) is a positive result from the algorithm where the instance is a member of the class. A *false positive* (fp) is a positive result from the algorithm where the instance is not a member of the class. A *true negative* (tn) is a negative result from the algorithm where the instance is not a member of the class. A *false negative* (fn) is a negative result from the algorithm where the instance is a member of the class.

Accuracy measures the percentage of instances that were correctly labelled by the system.

$$accuracy = \frac{tp + tn}{tp + fp + tn + fn} \quad . \tag{2.1}$$

This measure fails to capture the impact of the false positives and the false negatives. Two additional metrics called precision and recall attempt to correct for this bias. Precision is the ratio of the number true positives over the number of instances identified as positive and recall is the ratio of the number of true positives to the number of instances which were actually positive.

$$precision = \frac{tp}{tp + fp} \quad . \tag{2.2}$$

$$recall = \frac{tp}{tp + fn} \quad . \tag{2.3}$$

In an attempt to balance the two criteria, a combination is proposed [42]. The combination is known as the F measure. By equally weighting the combination, the following equation is defined as,

$$F = \frac{2 * precision * recall}{precision + recall} \quad . \tag{2.4}$$

Chapter 3

NHL Playoff Qualification and Elimination Problems

As a season progresses, sports fans become intensely focused on the playoff race and the position of their team in the standings. Sports sections of major newspapers publish the results of the games and announce when teams have qualified for the playoffs and when they have been eliminated (e.g. The Globe and Mail [26]). However, the newspapers use a heuristic measure for determining when teams have qualified for or been eliminated from the playoffs and announcements are sometimes not made until several days after the team has clinched or been eliminated. Since fans are interested in knowing when their team has clinched as early as possible, the exact answer is of more interest as the heuristic answer may not give precise results.

However, if the team has not clinched a playoff spot, this method provides no information about how close a team is to earning a playoff position. The problem of determining how close a team is to clinching a playoff spot can be modelled as an optimization problem that determines the minimum number of points that is necessary to guarantee a spot. This bound on the number of points can also be used to determine when a team has no guarantee of making the playoffs and when a team has lost a crucial game and left destiny in the hands of another team. These bounds give both fans and the managers of the teams additional information. For a coach, there is the additional benefit of knowing which games must be won so they can rest injured players before the playoffs. These factors are interesting to hockey fans and can be generalized to other sports with playoff structures, such as baseball and basketball. From a computational point of view, the qualification and elimination problems addressed here are NP-Hard problems and an efficient solution is not known to exist [45, 27].

In this chapter, a hybrid constraint programming and enumeration method is proposed to exactly solve qualification and enumeration problems for the National Hockey League

(NHL) playoffs[1]. Solutions to both the qualification and elimination problems use a phased strategy that solves enumerated sub-problems with network flows and constraint programming. Symmetry and dominance constraints are added to the model to improve efficiency.

Instances from the 2005-06 and 2006-07 seasons were experimentally evaluated using the solver. For these seasons, qualification of teams was shown up to five days earlier than the Globe and Mail [26]. All of the different scoring models that have been used by the NHL as well as the model used in the hockey tournament of the 2010 Olympic Winter Games were tested using the solver. The NHL tried the various scoring models to increase the competitiveness of the league though there exists no data on how these changes affected the clinching dates. In this thesis, it is shown that adjusting the scoring model can change the average qualification date by more than two days but the results vary significantly from year to year. Interestingly, the Toronto Maple Leafs would have qualified for the playoffs under any other scoring model than that used by the NHL in 2006-07 season. Another example where the scoring model mattered was the Edmonton Oilers in 2005-06 who were Western Conference champions but would not have made the playoffs if the scoring model from the Olympics had been used.

The solver proposed in this work can determine the minimum number of points for a given team, at any point in the season, within ten minutes and, for dates near the end of the season, in seconds. Equivalently, the minimum number of points needed to have any chance of making the playoffs can be determined in a similar fashion. In sports, analysts, reporters and coaches often refer to "must win" games. The method used in this paper can identify games where losing that game, the team puts its playoff aspirations into the hands of its opponents. While this does not mean the team will not qualify for the playoffs, it does mean that the team cannot guarantee a playoff spot. Nine teams in the 2006-07 season are identified that lost one of these "must win" games and found themselves in a position to earn a playoff spot again only through the actions of their opponents. Several teams experienced this phenomenon four times during the season.

This chapter first discusses some of the related work that exists in the literature. For those unfamiliar with the NHL, the NHL system is discussed as some background is necessary to understand the rest of the chapter. An introduction to the formal problem definition and some notation that will be used in the remainder of the chapter follows. A phased solver relying on the properties of tie-breaking constraints is proposed. Enumeration and bounding techniques are proposed to improve the efficiency of the solver. Lastly, some experimental results on real world NHL instances are presented.

[1]Portions of this work have previously been published in [57, 58, 59].

3.1 Related Work

The problem of determining when a sports team has mathematically clinched a playoff spot has been well studied for several sports, including baseball [63, 55, 69, 1] and soccer [54]. The problem is known as a *winner determination problem*. Schwartz [63] first looked at this type of problem algorithmically for the historical baseball problem. Two optimization variants of winner determination problems are discussed in this thesis.

Definition 3.1 (Playoff Qualification Problem)**.** *Given a remaining schedule of games left to play, the results up to a given point of the season—i.e. points and wins earned by teams so far—and a distinguished team t_q, the* playoff qualification problem *is to determine the minimum number of points needed by t_q such that if they earn that number of points there exists no scenario, i.e. a completion of the remaining games, such that t_q does not qualify for the playoffs.*

Definition 3.2 (Playoff Elimination Problem)**.** *Given a remaining schedule of games left to play, the results up to a given point of the season—i.e. points and wins earned by teams so far—and a distinguished team t_q, the* playoff elimination problem *is to determine the minimum number of points such that if t_q earns that number of points there exists at least one scenario where t_q earns a playoff spot.*

Playoff qualification criteria typically either select the top m teams or select the division leaders and a fixed number of extra teams, called wild card teams. These problems are known to be NP-Hard in general when there are m spots, where $m > 2$, that make the playoffs or when there is at least one wild card spot and multiple divisions [45, 27]. By restriction, problems solved using the current playoff structure of the NHL are also NP-Hard as the NHL playoffs have multiple divisions and wild cards and eight teams making the playoffs.

Approaches for solving the playoff qualification and elimination problems have been proposed for the Brazilian football championship [54] and for Major League Baseball [1]; both approaches use integer programming. Unlike hockey, these sports either have a simpler scoring model or a simpler playoff qualification method. Robinson [55] suggests a model for determining the number of points needed to clinch a playoff spot in the National Basketball Association and solves the model using integer programming techniques. Robinson [55] also suggested a model for the NHL but his model did not allow for wild card teams or tie-breaking conditions. Gusfield and Martel [27] put forth a method for calculating bounds on the conditions when a team has been eliminated from baseball playoffs but their method only works for a single wild card team and a simple win-loss scoring model. The solver in this chapter uses a similar method but differs in approach as there are multiple wild card teams in the NHL. Cheng and Steffy [16] looked at the problem of determining qualification for the NHL using an integer programming model but they found that they could not solve the model when secondary and tertiary tie-breaking rules were included.

Wayne [69] introduced the concept of a lower bound that could be used to determine whether or not a team was eliminated from the playoffs. Specifically, he introduced a lower bound on the minimum number of points needed to possibly earn a playoff spot. Gusfield and Martel [27] show how this idea can be extended to include a single wild card team and multiple division leaders. In this chapter, the existence of an upper bound which represents the minimum number of points needed to guarantee a playoff spot is discussed. Again, the problem in this chapter corresponds to the more complicated case where there are multiple wild cards.

Schwartz [63] showed in 1966 that the winner determination problem, could be solved in polynomial time for a win-loss scoring model using network flows. The basic method described by Schwartz [63] uses a flow network to partition remaining games to individual teams. If there exists a feasible flow, which represents an assignment of wins, then there exists a scenario where the distinguished team earns the most points and they have not be eliminated. Kern and Paulusma [34] showed that the same approach could be used for other scoring models if the scoring model is normalized. However, Kern and Paulusma make a strong assumption about the play of the distinguished team t_q as it is assumed t_q wins or loses every remaining game, depending on whether elimination or qualification is discussed, respectively. This is not feasible in this work as the assumption does not hold true in optimization problems where the points of t_q may not be at either extreme. As well, Kern and Paulusma's model does not explicitly state how to deal with the factors removed during normalization as they must be reincorporated into the model when dealing with tie-breaking conditions.

3.2 The NHL Playoff System

Since the last expansion of the league in 2000, the NHL has consisted of thirty teams arranged into two conferences, East and West, each of fifteen teams. Each conference is composed of three divisions with five teams each. Every team in the NHL plays 82 games with 41 home games and 41 away games. These games are spread unevenly with the highest number of games being played against teams in their own division, then their own conference and then finally the opposite conference. Currently teams play six games against each team in their division, four games against teams in their conference but not their division and one or two games against teams in the opposite conference. Each game in the NHL consists of sixty minutes of regulation time split into three periods of twenty minutes each. If the game is tied at the end of regulation time, a shorter overtime period of five minutes is played, which is sudden death i.e. ends when a goal is scored, and, if

Table 3.1: The scoring models that have been used by the NHL. The possible outcomes of an NHL game between two teams (i, j) are A) j wins in regulation time, B) j wins in overtime, C) tied game, D) i wins in overtime and E) i wins in regulation time. The table shows the points awarded in each of the possible outcomes for a given scoring model. If an entry is blank then the outcome is not possible under the scoring model.

Scoring Model		A	B	C	D	E	
Historic Era	{	$(0,2)$		$(1,1)$		$(2,0)$	}
Overtime	{	$(0,2)$	$(0,2)$	$(1,1)$	$(2,0)$	$(2,0)$	}
Extra Point	{	$(0,2)$	$(1,2)$	$(1,1)$	$(2,1)$	$(2,0)$	}
Shootout	{	$(0,2)$	$(1,2)$		$(2,1)$	$(2,0)$	}
Proposed	{	$(0,3)$	$(1,2)$		$(2,1)$	$(3,0)$	}

the score remains tied after overtime, a shootout is conducted, which must conclude with a winner. A team makes the playoffs if they are a division leader or one of the top five teams that are not division leaders in their conference. The top five teams in a conference who failed to win their conference are known as wild card teams. There are 16 teams in the playoffs.

Like many North American sports, an NHL game must end in a win or loss. However, the NHL has a unique scoring system as there are points awarded for reaching overtime even if a team does not win the game. If the game ends during regulation time then the winner of the game is awarded two points and the loser earns no points. If, however, the game ends either during the overtime period or the shootout then the winner still earns two points but the loser earns a single point in consolation.

The NHL has used several different scoring models over the years. A *scoring model* is defined as a set of tuples defining the possible outcomes, and subsequent reward, of the games. Each of the NHL scoring models can be viewed in Table 3.1. Referring to Table 3.1, the current scoring model is called the Shootout Model.

In the NHL, teams are placed in the standings by the number of points, for both divisional and overall standings. However, it is possible to have two teams with the same number of points. The NHL uses three different tie-breaking measures. The first tie-breaking measure is to compare the number of wins by each team. If the teams are still tied, the number of points earned against only those teams that are tied are compared.

There is a third measure, the total number of goals scored in the entire season, used by the NHL but not included in this work as it is impossible to determine beforehand how many goals will be scored in an NHL game. Instead, it is assumed that in the case of qualification that a team has not qualified if they need to win via the third tie breaker.

Conversely, in the case of elimination, it is assumed that a team has a chance to qualify if they could win via the third tie breaker.

3.3 A Motivating Example

Before proceeding with a more rigorous description of the various mechanisms used to solve these problems, an illustrative example is introduced that will be used throughout the chapter. Since a full scale problem using all of the teams in the league is too large to be effective as an example, a hypothetical six team league is constructed where teams play each other exactly four times, for a total of 20 games each. For simplicity, divisions are not introduced in this example but are discussed later in the chapter. In the example, four of our six teams make the playoffs. To illustrate the concept, our example contains a varying number of games played and games remaining against various teams. The schedule for the hypothetical league is constructed so that every team plays on each game day. Note that this is not necessarily the case in the NHL or virtually any other professional sports league and is done for clarity as when the games are played does not actually affect the mechanism by which problems are solved.

From the NHL, six teams were selected, Boston, Chicago, Detroit, Montreal, New York and Toronto, and a round robin tournament with four identical rounds was generated. The schedule can be seen in Figure 3.1. In the example, the first thirteen games of the season have been played and each team has seven games remaining. Again, the symmetry in terms of the number of games remaining is simply for clarity and simplicity and is not required by the techniques. From the results of the games, the standings at that point in the season can be constructed (see Table 3.2). The games remaining between each pair of teams is summarized in Table 3.3.

For the remainder of this chapter, this example is used to discuss how the various mechanisms of the solver work and why the mechanisms are implemented. At each point in the chapter, the fate of New York and the steps in determining how many points New York needs to clinch a playoff is discussed.

3.4 Basic Models

To formally specify the model, certain concepts and notations must first be introduced. The set of teams in the NHL is denoted $T = \{t_1, \ldots, t_n\}$. Let C_i be the set of teams in the conference and D_i be the set of teams in the division to which team t_i belongs. A scenario S is a completion of the schedule from current date of the season to the end of the season by assigning wins, losses, and overtime losses to the games scheduled after d_0.

30	31	1 M:0,T:2 NY:2,B:1 D:2,C:0	2 T:2,B:1 M:1,D:2 NY:2,C:0	3	4 T:2,NY:0 M:2,C:0 B:2,D:1	5 T:1,C:2 M:2,B:1 NY:0,D:2
6	7 T:2,D:0 NY:2,M:0 B:1,C:2	8 M:1,T:2 NY:2,B:0 D:1,C:2	9	10 T:0,B:2 M:2,D:1 NY:1,C:2	11	12 T:0,NY:2 M:2,C:0 B:1,D:2
13 T:0,C:2 M:2,B:1 NY:2,D:0	14	15 T:1,D:2 NY:2,M:0 B:2,C:0	16 M:2,T:1 NY:2,B:0 D:2,C:0	17	18 T:2,B:0 M:2,D:0 NY:2,C:0	19 T:2,NY:0 M:2,C:0 B:2,D:1
20	21 T v C M v B NY v D	22 T v D NY v M B v C	23	24 M v T NY v B D v C	25	26 T v B M v D NY v C
27 T v NY M v C B v D	28	29 T v C M v B NY v D	30 T v D NY v M B v C	1	2	3

Figure 3.1: An example schedule for six teams where each team plays each other team four times for a total of sixty of games. Thirteen of each team's games have been played and the results, in terms of points, are noted against the played games. The remaining seven games that each team must still play are listed with their scheduled opponents.

For every team t_i and opponent t_j, there exists a win variable, w_{ij}, and overtime loss variable, ol_{ij}, which represent the wins and overtime losses earned at the end of the season by t_i over t_j. For every team t_i and opponent t_j, constants W_{ij} and OL_{ij} represent the number of wins and overtime losses earned up to the current date of the season. The points, p_{ij} earned by the team t_i against an opponent t_j at the end of the season is the weighted sum of the wins, worth two points, and the overtime losses, worth a single point, notated $p_{ij} = 2w_{ij} + ol_{ij}$. Let G_{ij} be the constant representing the number of games remaining for a team t_i against a team t_j at the current date and $G_i = \sum_j G_{ij}$ is the sum of games remaining for team t_i against all teams. The total points earned at the end of the season by team t_i, p_i, is the sum of points earned against all opponents, $p_i = \sum_j (p_{ij})$, and the constant P_i represents the points earned by t_i up to the current date of the season. Let TB_i, the *tie-breaking set*, be the set of all of the teams tied with team t_i including t_i in both points and wins at the end of the season.

For each team t_i, let mpp_i be the maximum possible points that could be earned by t_i if they won all of their remaining games. Given a subset of teams $T' \subseteq T$, let $\max(T')$ be the maximum points over all teams in T' at the end of the season and let $\min(T')$ be the minimum points over all teams T' at the end of season.

A team only qualifies for a playoff spot if they are a division leader or a wild card team. A *division leader* is the team t_i that has the maximum number of points at the end of the

Table 3.2: The standings of the hypothetical league after the first thirteen games of the schedule have been played. Each team is awarded 2 points for each win, 0 points for each loss and 1 point for each overtime loss.

Team	Games Remaining	Games Remaining	Wins	Losses	Overtime Losses	Points
New York	13	7	9	3	1	19
Montreal	13	7	8	3	2	18
Toronto	13	7	7	3	3	17
Detroit	13	7	6	3	4	16
Boston	13	7	4	3	6	14
Chicago	13	7	5	8	0	10

Table 3.3: The number of games remaining for each team against each opponent after the thirteen completed games in the example.

Teams	Boston	Chicago	Detroit	Montreal	New York	Toronto
Boston	—	2	1	2	1	1
Chicago	2	–	1	1	1	2
Detroit	1	1	–	1	2	2
Montreal	2	1	1	–	2	1
New York	1	1	2	2	–	1
Toronto	1	2	2	1	1	–

season within their own division (i.e. $p_i = \max(D_i)$) and has better tie breakers than any team with equivalent points in their division at the end of the season. A *wild card team* is any team t_i that is not a division leader but has a p_i greater (or equal with better tie breakers) than at least seven other teams in C_i that are also not division leaders. Note that the number of teams, in this case seven, is simply the subtraction of the number of division leaders and wild card spots from the total number of teams in the conference.

A team has *qualified* for the playoffs when the number of points needed to guarantee a playoff spot is zero. A team controls their own destiny as long as the number of points needed to guarantee a playoff spot needed does not exceed the maximum points possible, mpp_i. A team has been *eliminated* when the number of points needed to possibly qualify for a playoff spot exceeds the maximum number of points possible.

The basic models for the qualification and elimination problem, discussed below, represent all of the necessary constraints for the two problems. Note that there is many ways to model these problems and this is just one, relatively simple, model for the problem.

The basic model of the NHL Qualification problem is a combination of six constraints (Constraints 3.1, 3.2, 3.3, 3.4, 3.5, 3.6, and 3.7) where some are more important than others. Following the example of Ribeiro and Urrutia [54], the problem of finding the minimum points necessary for a team to qualify is converted into the problem of finding the maximum number of points where there exists a situation where the team could have been eliminated. This is because it is often easier to find a solution when one exists than prove that there does not exist a solution. Therefore, the number of points needed to clinch a playoff spot is in fact one greater than the result returned from this model. Constraint 3.1 denotes that the problem is a maximization problem over the number of points earned by the team under consideration. Constraint 3.2 requires that the number of total wins between two teams equals the number of games they played. Constraint 3.2 enforces the constraint that each game must end in a winner. Constraint 3.3 states that the number of wins and overtime losses in any completed scenario must be less than or equal to the sum of the wins already earned, the overtime losses already earned and the total number of games remaining to be played. This ensures that each team earns only one win or one overtime loss for each game played. Constraint 3.4 formally defines the tie-breaking sets in terms of the model variables. The most important and most complicated constraint, Constraint 3.5, states in general that a team must be better on points, tie breakers or division leader status to be better than t_q. The first three terms of Constraint 3.5 correspond to the tie-breaking constraints. The fourth is a compound term that says that regardless of how well a team does against t_q as long as the team is the top team in their division then they qualify. It is important to note that this constraint is quadratic in nature as the third term is a sum over a set variable. Constraint 3.6 enforces that there must be at least eight teams, the number of playoff spots, that are better than the team under consideration. Constraint 3.7 ensures that there is at least one team which is better than the team under consideration and belongs to their division. This ensures that the team under consideration is not themselves a division leader.

An indicator variable b_i is introduced which is 1 if and only if t_i has more points or better tie breakers than t_q or is a division leader. If there exists eight or more teams where this is true then team t_q needs $\max p_q + 1$ points to guarantee a playoff spot. The problem with this model is that the core of the model is a combination of implication and disjunction constraints and such models can be difficult to solve. However, like other real world problems, the NHL is of bounded size and, therefore, it is sometimes practical to enumerate some of the variables or constraints of the model. In the next sections, the use of enumeration and decomposition to find solutions to models with Constraint 3.5 is discussed.

$$\max p_q \quad , \tag{3.1}$$

$$\forall i,j,i<j \qquad w_{ij} + w_{ji} = W_{ij} + W_{ji} + G_{ij} \quad , \tag{3.2}$$

$$\forall i,j,i<j \qquad w_{ij} + ol_{ij} \leq W_{ij} + OL_{ij} + G_{ij} \quad , \tag{3.3}$$

$$\forall i,j \qquad j \in TB_i \Leftrightarrow p_i = p_j \wedge w_i = w_j \quad , \tag{3.4}$$

$$\forall i \qquad b_i = 1 \tag{3.5}$$

$$\Leftrightarrow \qquad (p_i > p_q) \tag{3.5a}$$

$$\vee \qquad (p_i = p_q \wedge w_i > w_q) \tag{3.5b}$$

$$\vee \qquad \left(p_i = p_q \wedge w_i = w_q \wedge \sum_{t_j \in TB_q} p_{ij} > \sum_{t_j \in TB_q} p_{qj}\right) \tag{3.5c}$$

$$\begin{aligned} \vee \qquad & \Bigg[\forall_{t_d \in D_i} (p_i > p_d) \\ & \vee (p_i = p_d \wedge w_i > w_d) \\ & \vee \left(p_i = p_d \wedge w_i = w_d \wedge \sum_{t_j \in TB_i \wedge t_j \in D_i} p_{ij} > \sum_{t_j \in TB_i \wedge t_j \in D_i} p_{dj}\right)\Bigg] \quad , \end{aligned} \tag{3.5d}$$

$$\sum_{t_i \in C_q} b_i \geq 8 \quad , \tag{3.6}$$

$$\begin{aligned} \exists i \qquad & t_i \neq t_q \wedge t_i \in D_q \\ \wedge \qquad & \Bigg[\forall_{t_d \in D_q} (p_i > p_d) \\ & \vee (p_i = p_d \wedge w_i > w_d) \\ & \vee \left(p_i = p_d \wedge w_i = w_d \wedge \sum_{t_j \in TB_i \wedge t_j \in D_q} p_{ij} > \sum_{t_j \in TB_i \wedge t_j \in D_q} p_{dj}\right)\Bigg] \quad . \end{aligned} \tag{3.7}$$

The NHL Elimination Problem has a similar model where the solution is the minimum number of points where there are at least seven teams that do not do better than the distinguished teams. In other words, the goal is to find the solution where the team under consideration could do no worse and still make the playoffs. The constraints described in Constraints 3.9, 3.10 and 3.11 are identical to Constraints 3.2, 3.3 and 3.4. Constraint 3.12 says that a team is worse than t_q if they have less points or equal points and worse tie-breakers than t_q or is a division leader. Constraint 3.13 says there is a solution where t_q qualifies if there are seven teams worse than t_q or t_q is a division leader.

$$\min p_q \quad , \tag{3.8}$$

$$\forall_{i,j,i<j} \qquad w_{ij} + w_{ji} = W_{ij} + W_{ji} + G_{ij} \quad , \tag{3.9}$$

$$\forall_{i,j,i<j} \qquad w_{ij} + ol_{ij} \leq W_{ij} + OL_{ij} + G_{ij} \quad , \tag{3.10}$$

$$\forall i,j \qquad j \in TB_i \Leftrightarrow p_i = p_j \wedge w_i = w_j \quad , \tag{3.11}$$

$$\forall_i \qquad \omega_i = 1 \tag{3.12}$$

$$\Leftrightarrow \qquad \Big[(p_i < p_q) \tag{3.12a}$$

$$\vee \qquad (p_i = p_q \wedge w_i < w_q) \tag{3.12b}$$

$$\vee \qquad \left(p_i = p_q \wedge w_i = w_q \wedge \sum_{t_j \in TB_q} p_{ij} < \sum_{t_j \in TB_q} p_{qj} \right) \Big] \tag{3.12c}$$

$$\begin{aligned} \wedge \qquad & \Big[\exists_{t_d \in D_i} (p_i < p_d) \\ & \vee (p_i = p_d \wedge w_i < w_d) \\ & \vee \left(p_i = p_d \wedge w_i = w_d \wedge \sum_{t_j \in TB_i \wedge t_j \in D_i} p_{ij} < \sum_{t_j \in TB_i \wedge t_j \in D_i} p_{dj} \right) \Big] \quad , \end{aligned} \tag{3.12d}$$

$$\begin{aligned} & \sum_{t_i \in C_q} \omega_i \geq 7 \\ \vee \qquad & \Big[\forall_{t_d \in D_q} (p_q > p_d) \\ \vee \qquad & (p_q = p_d \wedge w_q > w_d) \\ \vee \qquad & \left(p_q = p_d \wedge w_q = w_d \wedge \sum_{t_j \in TB_q \wedge t_j \in D_q} p_{qj} > \sum_{t_j \in TB_q \wedge t_j \in D_q} p_{dj} \right) \Big] \quad . \end{aligned} \tag{3.13}$$

3.5 Solution Overview

Constraint 3.5 is a combination of a logical implication and a disjunction where even maintaining arc consistency may prune very few domains until late in the search. However, mutually exclusive disjunctions provide a simple method for decomposing the model as only one of the conditions can be true at any given time. One method for dealing with mutually exclusive disjunction is through branching on exclusive choices in the disjunction as described by Van Hentenryck [29, pg. 169]. The approach here, of enumerating and using different solvers, differs because the feasibility of combinations of disjunctions can be easily checked and it is possible to determine when a problem is hard and can be reserved until later, if needed. Observe that in Constraint 3.5 the first three conditions (Constraints 3.5a, 3.5b, 3.5c) have a mutually exclusive structure,

$$\begin{aligned} & (p_i > p_q) \vee (p_i = p_q \wedge w_i > w_q) \\ \vee \quad & \left(p_i = p_q \wedge w_i > w_q \wedge \sum_{t_j \in TB_q} p_{ij} > \sum_{t_j \in TB_q} p_{qj} \right) \quad . \end{aligned}$$

Each constraint naturally excludes the others being true and thus provides a simple decomposition as each term in the constraint being true means the other terms cannot be true. Also, these terms form a tie-breaking constraint which means that the relaxed version of the first, where equality is allowed, would include all solutions to the second and third and the relaxed version of the second with equality would include the solutions to the third.

These observations allow for a phased strategy where each disjunction is added only if necessary. This is efficient in this case as the tie-breaking constraints are often not necessary and it is often the case that delaying the enforcement of the constraint in its full complexity means that it never has to be completely solved. Algorithm 3.1 gives the basic algorithm that is used to solve the problems.

The first phase only enforces Constraint 3.5a and not Constraints 3.5b and 3.5c which corresponds to determining the minimum number of points where t_q guarantees a playoff spot only via points. One of the techniques used in this phase is to enumerate all of the possible teams that could occupy the eight playoff positions. Determining the maximum number of points that the weakest team filling the spots could earn gives a bound, for that set, on the number of points that t_q would need to earn. If this number of points, for all sets, is less than P_q, the points currently earned by t_q so far, then t_q would not need to earn anymore points and they would have guaranteed a playoff spot. If the weakest team in one of the sets earned more points than t_q could possibly earn (mpp_q) then t_q cannot guarantee a playoff spot. In all other situations, there is a maximum number of points for each of the weakest teams and, as described in more detail in Section 3.6, only the sets with the highest point value are kept. This is the best pruning that can be achieved at this level. Since t_q could possibly qualify, given further tie-breaking, at the bound calculated, each set remaining after pruning along with the highest point bound $\mathbf{p}$ are given to the second phase of the solver.

The second phase of the solver enforces Constraints 3.5a and 3.5b but not Constraint 3.5c which corresponds to determining the minimum number of points where t_q guarantees a playoff spot using points and wins. Given the sets from the first phase and the point bound $\mathbf{p}$, it is determined if there exists a set of teams such that there exists a scenario where each team has more than $\mathbf{p}$ points or exactly $\mathbf{p}$ points and more wins when t_q has $\mathbf{p}$ points. If this exists then t_q would need one more point to earn more points than the teams in that set and, since $\mathbf{p}$ is the maximum over all sets, all sets. If t_q cannot earn that extra point, when $\mathbf{p} \geq mpp_q$, then there exists a scenario where t_q cannot qualify. If no such set exists, then it is necessary to determine if any of the sets could have every team have more points or equal points and equal or more wins. If not, then t_q would guarantee with $\mathbf{p}$ points and, if so, then further tie-breaking would be necessary. By recording all of the possible win values where teams could be tied, it is easier to determine a solution in the third phase. Therefore for each set remaining, triples are constructed with a set, the point bound and a win bound, $\mathbf{w}$, that is feasible for the set.

The third phase solves the complete model. If there is a solution then t_q needs $\mathbf{p} + 1$ points to guarantee. If $\mathbf{p} \geq mpp_q$ then t_q would not be able to guarantee a playoff spot. If there is no solution to the model then t_q needs $\mathbf{p}$ points to qualify.

A variety of different techniques are used to solve the problems. In each stage, there is some enumeration and feasible flow calculations. In the third stage, constraint programming is used to solve the complete model. Each of these techniques are expanded in more detail in Sections 3.6, 3.7 and 3.8.

Much of the same mechanisms can be used for the playoff elimination problem. The goal is to search for any set of seven teams to occupy the spots below t_q. If this set of teams exist or if t_q is a division leader at a given point bound then t_q can possibly qualify for the playoffs. Using a similar bounding technique, a tight bound is obtained in a similar manner and the same phased solving methods can be applied as the same tie-breaking conditions apply in both situations. One difference between the elimination version of the problem and qualification version is that we are looking for solutions at the bound instead of proving that none exist, which is beneficial as it is often much more efficient to prove the existence of a solution.

3.6 The First Phase

In this section, the mechanisms for solving the first phase of the solver are described. In the previous section, the enumeration of sets of teams that could potentially be the division leaders and wild card spots is mentioned. In this section, some formal justification for this enumeration and some discussion of how the enumeration can be used to satisfy the logical equivalence of Constraint 3.5 while also satisfying Constraint 3.6. Given the sets of teams,

```
Algorithm:Solver(Games Remaining, Results, $t_q$)
input  : The schedule of games remaining, the results of the season so far and a
         team $t_q$
output: The minimum number of points needed by $t_q$ to guarantee a playoff spot
// FIRST PHASE
ans1 ← For every possible set of division leaders and wild card teams, is the maximum number of
points earned by the weakest team in the set less than $P_q$, the current points of $t_q$?;
if ans1 = 'yes' then
    return $P_q$;
ans2 ← Does there exist a set of division leaders and wild card teams where the maximum number
of points earned by the weakest team in the set is less than or equal to $mpp_i$?;
if ans2 = 'no' then
    return cannot guarantee;
enumerate the sets of teams such that ans1 = 'no' and ans2 = 'yes';
find the bounds for each set;
prune sets without the highest bound $\mathbf{p}$;
// SECOND PHASE
ans3 ← For every set of division leaders and wild card teams, is there a scenario where every team
in the set has more points or equal points and more wins than $t_q$ given that $p_q = \mathbf{p}$?;
if ans3 = 'yes' and $\mathbf{p} < mpp_q$ then
    return $\mathbf{p} + 1$;
else if ans3 = 'yes' and $\mathbf{p} \geq mpp_q$ then
    return cannot guarantee;
ans4 ← For every set of division leaders and wild card teams, is there a scenario where every team
in the set has more points or equal points and equal or more wins than $t_q$ given that $p_q = \mathbf{p}$?;
if ans4 = 'no' then
    return $\mathbf{p}$;
enumerate the sets of teams such that ans3 = 'no' and ans4 = 'yes';
determine the win bound $\mathbf{w}$ for each set;
// THIRD PHASE
ans5 ← For every set of division leaders and wild card teams, is there a scenario where every team
earns a playoff spot given that $t_q$ has $\mathbf{p}$ points and $\mathbf{w}$ wins?;
if ans5 = 'yes' and $\mathbf{p} < mpp_q$ then
    return $\mathbf{p} + 1$;
else if ans5 = 'yes' and $\mathbf{p} \geq mpp_q$ then
    return cannot guarantee;
else
    return $\mathbf{p}$;
```

Algorithm 3.1: The overview of the solver phases. Each phase corresponds to solving the problem with increasing levels of tie-breaking. The first phase solves the problems where only the points of the teams are considered. If teams may still be tied on points, the second phase solves the problem where points and wins are used to compare teams. If the teams are still tied then the third phase solves the problem where points, wins and points against tied teams are used to compare teams.

a bound on the number of points earned by the weakest team is developed. Combined with the enumeration of all sets, the bounding of the sets provides a tight lower bound on the actual value of the complete problem. Given the enumeration and bounding, it is simple to answer the two necessary questions for the first phase. If the bound $\mathbf{p}$ is less than P_q then t_q has already qualified and if $\mathbf{p}$ is greater than mpp_q then guaranteeing qualification is not possible.

3.6.1 Enumerating the Set of Implication Constraints

Implication constraints are widely used in constraint models. Examples of applications that include implication constraints include lattice protein folding [8], configuration problems [33], telecommunication feature subscription [37], pipeline planning and scheduling [46], and the travelling salesman problem with time windows [51]. However, while they easily capture the constraints inherent in many applications, they provide relatively weak propagation [39].

Enumeration is a common operations research and constraint programming technique for decomposing a problem into more manageable sub-problems. Often, the goal is to enumerate the values of specific variables to decompose the constraint graph or propagate a singleton value. This differs from the goal here which is to allow constraints to be posted earlier to increase the propagation available to the solver. Examples of applications which use enumeration as decomposition technique include sports scheduling problems [49], instruction scheduling [41], and diagnostics [61].

Rymon introduced a technique for enumerating the power set of possible sets systematically in a best first fashion [61]. This is similar to the technique used here but again the enumeration used by Rymon was on variable values whereas, in this work, it is on constraint implications which does not necessarily result in any singleton variable propagation but rather allows the implied constraint to be posted earlier than in the simple model.

Enumerating implication constraints can be applied to Constraint 3.5 for the NHL problems. Given the limited size of an NHL conference, it is possible to enumerate all of the different ways that the indicator variables b_i could be set in the model. Examining one side of the logical equivalence in Constraint 3.5, it is possible to derive,

$$\forall_i b_i \Rightarrow (p_i > p_q) \vee (p_i = p_q \wedge \ldots \quad . \tag{3.14}$$

Constraint 3.5 is a constraint of logical equivalence and does not necessarily have the dominance described. However as long as Constraint 3.6 is satisfied, Constraint 3.14 can be substituted for Constraint 3.5 because as long as eight teams are better as specified in Constraint 3.6 then additional b_i variables do not necessarily need to be true.

An *Elimination Set*, E, is a set of eight teams from the same conference with at least one team from each division and does not include t_q. Each team $t_i \in E$ must either have $mpp_i > P_q$ or be the only team in E from a division D_i such that $D_i \neq D_q$.

Example 3.1 (Finding Elimination Sets). *Referring to the example in Section 3.3, the hypothetical league has four playoff positions. Hence, the elimination set has four teams. Given t_q = New York, the elimination sets are formed from the remaining five teams. Every subset of size four which can earn more points than the number of points New York currently has is a viable elimination set. In this case, every team can earn more points than New York currently has and the elimination sets are:*

- $\{Montreal, Toronto, Detroit, Boston\}$
- $\{Montreal, Toronto, Detroit, Chicago\}$
- $\{Montreal, Toronto, Boston, Chicago\}$
- $\{Montreal, Detroit, Boston, Chicago\}$
- $\{Toronto, Detroit, Boston, Chicago\}$

A *Qualification Set*, Q, is a set of seven teams from the same conference which does not include any team that has clinched a division leadership position and does not include t_q.

Example 3.2 (Finding Qualification Sets). *Referring to the example in Section 3.3, the hypothetical league has only six teams and four playoff spots. Therefore, a qualification set would have two teams. The only pairs in this example which are not qualification sets are the ones including New York. The qualification sets are:*

- $\{Montreal, Toronto\}$
- $\{Montreal, Detroit\}$
- $\{Montreal, Boston\}$
- $\{Montreal, Chicago\}$
- $\{Toronto, Detroit\}$
- $\{Toronto, Boston\}$
- $\{Toronto, Chicago\}$
- $\{Detroit, Boston\}$
- $\{Detroit, Chicago\}$
- $\{Boston, Chicago\}$

3.6.2 Calculating the Bound

It is well known that tight lower bounds are useful in solving combinatorial optimization problems. The most common use is in branch and bound search where lower and upper bounds are used to prune infeasible regions of the search space. Mixed integer programming relies heavily on this technique due to the existence of an easily obtainable polynomial relaxation, the LP relaxation. In a constraint programming context, an integer optimization function may be represented as a constrained integer whose domains may be pruned by good bounds.

Since the NHL Playoff Qualification problem is an optimization problem, not all of the feasible solutions lead to an optimal solution and bounding can be used to remove feasible solutions that are non-optimal. Before presenting the bounding technique used to find bounds on the elimination and qualification sets described in the previous section, two generic lemmas and some notation are introduced to help prove the correctness of the bounds on the sets.

Given a set of implication constraints of the form, $p_i \Rightarrow q_i$, where p_i is an indicator variable, i.e. a constraint where there is a single variable constrained to be a single value, and q_i is any constraint that does not include the variable in p_i and the ability to check the feasibility of all other constraints on the indicator variables, a generic dominance rule about implication constraints of this form can be created. An *assignment* of the indicator variables p_i is the set of indicator variables P_T which are set to true while all other indicator variables are set to false. For each indicator variable $p_i \in P_T$, there is a corresponding q_i constraint which must be satisfied in the model and let Q_T be the set of constraints that must be satisfied under an assignment P_T. Let $I(p_i)$ be the set of constraints that can be affected by fixing the value of an indicator variable p_i, directly or by chaining through other variables in the constraints and $I(P_T) = \bigcup_{p_i \in P_T} I(p_i)$ be the set of all constraints affected by setting a set of indicator variables P_T. A constraint c is said to be affected by chaining if there exists a sequence of constraints from c to a constraint containing the indicator variable p_i such that a constraint in the sequence shares at least one variable with its predecessor and successor in the sequence.

Lemma 3.1 (Superset Dominance of Implication Constraints)**.** *Given a set of implication constraints, $p_i \Rightarrow q_i$, and a set of other constraints in the model, if there exists an assignment of p_i variables, P_T, that yields a feasible solution to the model then any sub-assignment $P_T^- \subseteq P_T$ yields a feasible solution to the model as long as the constraints in $I(P_T)$ have a feasible solution given the assignment P_T^-.*

Proof. Assume that there is a feasible solution to the model given the assignment P_T. Now assume that the model is infeasible for some sub-assignment of P_T, P_T^-. This means that some constraint in the model that was not violated under the assignment P_T has now been violated under the assignment P_T^-. Since the implication constraints enforced by P_T^- are a subset of the implication constraints under the assignment P_T and the model is strictly more relaxed, the only constraints which could have caused the model to become infeasible are those in the set $I(P_T)$. $\square$

There exists another dominance in the set of indicator variables. If the constraints enforced by an assignment are infeasible then any set containing that assignment must also be infeasible.

Lemma 3.2 (Subset Dominance of Implication Constraints)**.** *If the set of constraints Q_T associated with an assignment P_T are infeasible, then any set P_T^+, such that $P_T \subseteq P_T^+$, enforces an infeasible set of constraints.*

Proof. Since the set of constraints Q_T^+ enforced by P_T^+ contains the infeasible subset of constraints Q_T, the entire set of constraints is infeasible. $\square$

Given Lemma 3.1 and Constraint 3.6, if there is a feasible solution with more than eight constraints enforced then there is a feasible solution with exactly eight implication constraints enforced. In this section, it is shown that an assignment can be bounded and that smaller feasible sets always have a solution value at least as large as a super set.

The bound of an elimination set, E, is the $\max(\min(E))$ under all scenarios S where either $p_q = \min(E)$ or $p_q = mpp_q$. The bound on the qualification set, Q, is the $\min(\max(Q))$ under all scenarios S where either $p_q = \max(Q)$ or $p_q = P_q$.

A bound is constructed by taking an assignment to the indicator variables and relaxing Constraint 3.14 to,

$$\forall_i b_i \Rightarrow (p_i \geq p_k) \quad . \tag{3.15}$$

Constraint 3.15 gives the relaxed form of the original constraint where only the first tie-breaking condition is considered. The solution to the relaxed problem with Constraint 3.15 is a lower bound on the original problem since the distinguished team may need one more point to actually break ties which are relaxed using (3.15). The following lemma formalizes this notion.

Lemma 3.3 (One More Point Lemma)**.** *The solution to the basic constraint model given an assignment P_T corresponding to an elimination set and where Constraint (3.14) is replaced by Constraint (3.15) is a tight lower bound on the original problem that is at most one less than the actual solution for the given set of indicator variables if a solution exists.*

Proof. The solution obtained by substituting the relaxed constraint finds the maximum value of p_q such that each team t_i where $b_i \in P_T$ has at least as many points as t_q. If t_q earns one more point then there must exist one team t_i where $b_i \in P_T$ that cannot simultaneously obtain $p_q + 1$ points along with t_q or the solution was not the maximum. Therefore, if t_q earns one more point than the relaxed bound, they necessarily qualify and could qualify at the lower bound if they are better on tie breaks. $\square$

Now, it is shown that a smaller assignment necessarily has a bound value that is at least as large as an assignment which sets every variable in the smaller assignment.

Lemma 3.4 (Smaller Subsets are Optimal). *Given two assignments, P_T^+ and P_T, such that $P_T \subseteq P_T^+$, the solution for the NHL qualification problem under the assignment P_T has a value that is at least as large as the value of the solution to the NHL qualification problem under the assignment P_T^+ as long as both models have a feasible solution.*

Proof. Assume there is a feasible solution to both models and that the solution under the model enforcing the constraints in Q_T has a value that is less than the value of the solution to the model enforcing the constraints in Q_T^+. However, the constraints in the model under the assignment P_T are a subset of the constraints in model under the assignment P_T^+. Therefore, any solution to the P_T^+ model is a solution to the P_T model as long as Constraint 3.6 is satisfied under P_T, which is given. Therefore, the solution to model under the assignment P_T must have a solution value at least as large as the solution value of the model under the assignment P_T^+. $\square$

Lemma 3.4 states that smaller sets are optimal if they are feasible and Lemma 3.2 shows that any superset of a small set is infeasible if the smaller set is infeasible. As a result, this means that it is sufficient to only look at the smallest sets that satisfy both the constraints on the indicator variables and the constraints on the rest of the model.

In a playoff qualification problem, those solutions where the distinguished team t_q may earn a playoff spot but earning one more point will surely earn them a playoff spot give the best bound. Adapting an idea by Brown [13], the problem is solved by calculating a sequence of feasible flows. The algorithm starts with an upper bound found via a relaxation. Given the upper bound, it is determined, via a feasible flow calculation, if every team in the elimination set could earn at least that many points. If not, the upper bound is reduced until a feasible solution is found.

In order to find the best bound, teams in the elimination set E earn as many points as possible. This means that every loss by a team in the elimination set is an overtime loss and teams in the elimination set are expected to win all of their games against teams that are not in the elimination set except against t_q. The points earned by a team t_i given the above criteria is formalized as,

$$p'_i = P_i + 2\sum_{j \notin C_i} g_{ij} + 2\sum_{j \notin E \cup \{t_q\}} g_{ij} + \sum_{j \in E \cup \{t_q\}} g_{ij} \ . \tag{3.16}$$

Equation 3.16 represents the sum of the points already earned (P_i), the wins against teams not in the set $E \cup \{t_q\}$ ($2\sum_{j \notin C_i} g_{ij} + 2\sum_{j \notin E \cup \{t_q\}} g_{ij}$) and one point each from games against teams in $E \cup \{t_q\}$ ($\sum_{j \in E \cup \{t_q\}} g_{ij}$). The preprocessing step is a valid dominance relation as the scenario with the maximum min (E) is desired and these steps either increase the points of a team in E or leave them the same while not affecting the maximum possible points of the teams in E.

Lemma 3.5 (Adjusted Points Lemma)**.** *A solution with points adjusted as in Equation 3.16 dominates all other solutions for a given elimination set E.*

Proof. Assume there exists a scenario where the bound value is greater than any bound where points are first adjusted as in Equation 3.16. This means that either one or more teams lost some games to the opposite conference, lost a game to a team not in $E \cup \{t_q\}$, or failed to earn an extra point when losing to a team that was in $E \cup \{t_q\}$. **(Case 1)** Assume there exists a team t_i which lost games to the opposite conference. Since the points earned against opposite conference teams do not affect any other team in $E \cup \{t_q\}$, t_i could win those games and the bound would either increase or stay the same. Therefore the bound is never decreased by winning against the opposite conference and for there to be a scenario where the bound is higher without the adjustment it must occur in the other two conditions. **(Case 2)** Assume there exist a team t_i that does not win all games that are not in the set of elimination teams or t_q. If this was not the case, then there exists games that t_i could win without affecting how many points the other teams in $E \cup \{t_q\}$ earn. As with the previous case the bound either increases or stays the same by winning those games and therefore the scenario where the bound is higher without adjustment must have occurred due to the non-adjustment of the third condition. **(Case 3)** Assume there exist a team that does not earn at least one point from any game that they participate in against teams in $E \cup \{t_q\}$. Team t_i need not win these games but, if they lose, the point bound would be no lower if they lost in overtime, earning one point. Therefore, the third and final criteria cannot cause the bound to be lowered and there is a contradiction. Therefore, any scenario which enforces the adjustment has a solution value at least as high as any scenario which does not adjust the points. □

The Relaxed Bound

To determine the initial value for the bounding procedure, a relaxation of the bound calculation is solved where the constraint that a specific number of games must be played between two teams is relaxed. Therefore, every game between teams in the elimination set are considered as a pool of games and are assigned to the worst team until the games are used up or the $\min_{i \in E}(mpp_i)$ is reached. As well, the constraint that t_q could reach the bound if it is not larger than mpp_q is relaxed. While this scenario is not necessarily a feasible solution, the relaxed bound is a good overestimate of the actual bound. Lemma 3.6 shows that the relaxed bound is an upper bound of the actual bound.

Lemma 3.6 (The Relaxed Bound)**.** *The relaxed bound is an upper bound of the actual bound.*

Proof. Proof by contradiction. Assume that there exists an actual bound which is higher than the relaxed bound given by the algorithm. This assumes that there exists some assignment of wins that is not possible under the relaxed bound. However, the relaxed bound is a valid relaxation since two constraints are removed and none are added. Therefore, the solution to the original problem must be contained in the set of solutions to the relaxed problem. Therefore, any bound achievable on the original problem is achievable under the relaxation. □

Algorithm 3.2 shows the steps of the algorithm. The maximum upper bound of the algorithm is the lowest mpp_i for all teams $t_i \in E$. Next, add all of the points described by Equation 3.16 as well as any points that could be earned by playing t_q. During this step, a count of the number of remaining games is kept. The teams are sorted by increasing point values. Finally, games are added from the pool of remaining games to the worst or set of equally worst teams until the maximum upper bound is reached or the pool contains no more games.

Example 3.3 (Calculating the Relaxed Bound)**.** *Referring to the example described in Section 3.3, the first elimination set described previously in Example 3.1 is Montreal, Toronto, Detroit and Boston. This leaves New York and Chicago out of the set. The maximum upper bound is 28 as Boston has the lowest maximum possible points, where* $mpp_B = 28$*. Figure 3.2a shows the current points of the teams within the set (See Table 3.2). Referring to Table 3.3, each team within the set plays 2 games against either New York or Chicago and one against the other. As well, each team within the set plays one game against two teams within the set and two games against the other team in the set. Figure 3.2b shows the points, two per game, added for the games outside the set and the points, one point per game, added for the games inside the set. The next step in the algorithm sorts the teams in ascending order as shown in Figure 3.2c. At this point, there are eight games remaining between the teams within the set. Boston has two less points than Detroit so two of the remaining games are assigned to Boston (See Figure 3.2d). Now Boston and Chicago have one less point than Toronto, so one game is added to both of those teams (See Figure 3.2e). With four games remaining, Boston, Chicago and Toronto each have*

```
Algorithm: RelaxedBound($T$, $E$, $g$)
input  : The set of teams $T$, the elimination set $E$ and the games remaining $g$
output: Returns the relaxed bound for $E$.
Initialize Points array to zero;
Max $\leftarrow$ the smallest $mpp_i$ of all teams in $E$;
// Adjust The Points
for $t \in E$ do
    Points[$i$] $\leftarrow P_i$;
    for $j \in T$ do
        if $j \in E$ then
            Points[$i$] $\leftarrow$ Points[$i$] $+ 1$;
            GamesRemaining $\leftarrow$ GamesRemaining $+ 1$;
        else
            Points[$i$] $\leftarrow$ Points[$i$] $+ 2$;
sort_ascending(Points);
current $\leftarrow$ Points[0]; // Initial Starting Points
GamesRemaining $\leftarrow$ GamesRemaining/2; // Due to Double Counting
// Assign The Remaining Games to the Weakest Teams
while GamesRemaining $> 0$ do
    if current $\geq$ Max then
        return Max;
    difference $\leftarrow$ the number of teams with points equal to current;
    if difference $<$ GamesRemaining then
        Increment Points by one for each team with points equal to current;
        current $\leftarrow$ current $+ 1$;
    else
        return current;
return current;
```

Algorithm 3.2: This algorithm describes the method by which a relaxed infeasible bound is calculated.

one point less than Montreal and one game each is assigned to those teams (See Figure 3.2f). There is a single game remaining but four games are required to increase the bound plus the maximum upper bound has been reached. All other elimination sets would have a bound of 24 because Chicago can earn at most 24 points and Chicago is a member of all of the other sets.

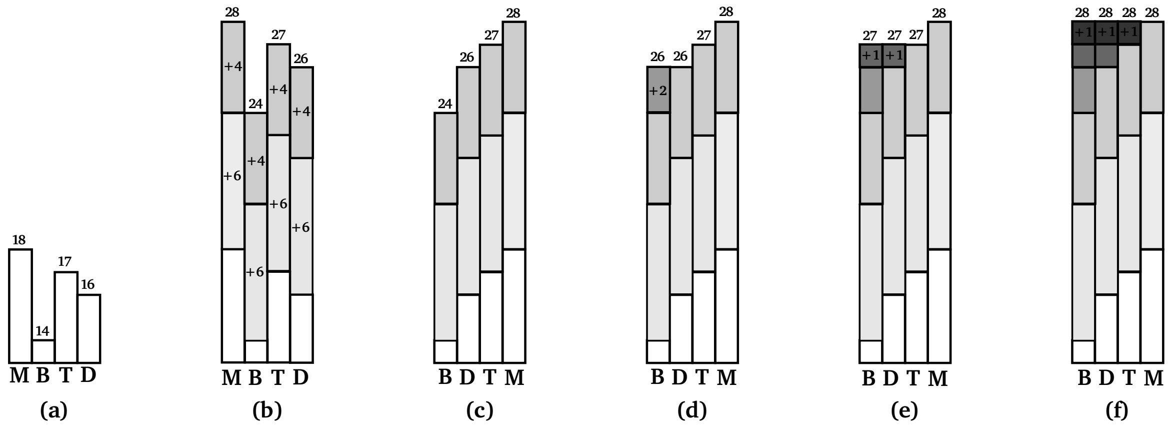


Figure 3.2: The application of the relaxed bound algorithm on the example. **(a)** The number of points earned by each team. **(b)** Added six points for each team for the three games they play against both New York and Chicago and four points for the four remaining games. **(c)** The sorted list of teams. **(d)** Two wins are added to Boston. **(e)** A single point each is added to Boston and Detroit. **(f)** Three points are added to the three weakest teams and the algorithm terminates with a single game remaining.

The Flow Network and the Bound

Once an upper bound is calculated by the relaxed bound algorithm, the bounding procedure must be applied (see Algorithm 3.3). The algorithm uses a flow network to model the problem of finding a bound for a given elimination set. Described in detail later, a flow network is constructed such that a feasible flow on the network ensures that there exists a scenario where each team in the set earns the number of points needed to reach a given bound. If the problem is infeasible, then the bound is reduced and the computation is repeated until a feasible solution is found.

Given a bound, every team in the elimination set and the team t_q needs to win a certain number of games to reach the bound. The *need*, n_i, of a team t_i is the difference between the bound and the adjusted points of t_i.

$$n_i = \max\left(bound - p'_i, 0\right), \tag{3.17}$$

where *bound* is the current lower bound on points and p'_i is defined in Equation 3.16.

Once the *needs* are known for a given elimination set, it is relatively simple to construct the flow network to solve the problem. A source node s and a sink node t are added. The bound on an elimination set requires that each team in the elimination set reaches the

```
**Algorithm:**Bound($E$)
input  : The elimination set $E$.
output: Returns bound on the elimination set $E$.
bound ← RelaxedBound(E, $t_q$);
repeat
    Needs ← CalculateNeeds($E$, $t_q$, bound);
    need ← $\sum_{i \in E} n_i$;
    G ← ConstructGraph(Needs);
    flow ← CalculateFlow(G);
    if flow < need then
        bound ← bound − 1;
until flow ≥ need ;
return bound
```

Algorithm 3.3: This algorithm shows the steps for calculating the bound for a given elimination set, E. First, calculate the relaxed bound as a starting point. From that starting point, generate, for each team, the number of points needed to reach the bound, denoting the set of *needs* as $Needs$. Then, check feasibility using a flow algorithm. If the flow meets the *needs*, return the bound. Otherwise, reduce the bound and iterate.

bound and that p_q is exactly the bound or, if the bound is too large, mpp_q, and there is no restriction on the points of the other teams. Therefore, the network must include all of the teams from the elimination set and t_q to ensure that each reaches the bound, if possible. A node for every matchup of teams from the set composed of the elimination set plus t_q is added and, in second column, a node for each team in the elimination set plus t_q. There is an arc from s to every node representing a matchup of two teams with an upper and lower bound capacity equal to the number of games that they play. From each node representing matchups, there are two outgoing edges, one for each team in the matchup, with a lower bound capacity of zero and an upper bound capacity equal to the number of games remaining between the two teams. The flow along the edges from the matchups represents an outcome of the games in some scenario. From each node representing a team, there is an edge to the sink node t with a lower bound capacity equal to the *need* of the node and an upper bound capacity equal to the number of games the team has remaining. One exception is that the edge from the node representing t_q to the sink has an upper and lower bound capacity equal to the need. A feasible flow on this network solves the bounding problem because it ensures that at least one team wins every game and that each team in the set meets their need to reach the point bound. An example of such a network can be found in Figure 3.3.

Example 3.4 (Calculating the Tight Lower Bound)**.** *Referring to the example described in Section 3.3, the bounding technique is applied to the first set of teams generated in Example*

3.1. Recall that in Example 3.3, the upper bound for this set is 28. Now a feasible bound must be calculated. First, the adjusted points are calculated as described in Equation 3.16. Subtracting from 28 the adjusted points of each team in the set as well as New York, the need for each team is determined. Now, the flow network in Figure 3.3 is constructed. A sink node s and a source node t are added. The first column of nodes represents the ten possible pairings of opponents between all of the teams in the set and New York. And the second row of nodes represents each team. Each pairing of opponents has some number of games to be played. For each, an edge is added from the source with an upper and lower bound capacity equal to the number of games to be played. For example, New York plays Montreal twice so the edge from the source to the New York-Montreal pairing is two. From the pairing nodes, there are two outgoing edges, one to each of the teams in the pairing. These have an upper bound capacity of the number of games to be played and a lower bound capacity of zero. In the case of the New York-Montreal pairing, there is an edge to New York and an edge to Montreal and both of them have an upper bound capacity of 2 and a lower bound capacity of zero. From each team node, an edge to the sink is added with the lower bound capacity being the need of that team to reach the bound and the upper bound capacity being the maximum number of games the team could win. Note that New York is reaching the bound exactly so its upper and lower bound capacities are equal. Montreal, alternatively, has a need, and hence a lower bound capacity, of two, but an upper bound capacity of six as members of the elimination are only constrained on their lower bound. Once the flow network has been constructed, a feasible flow algorithm is applied [2]. Figure 3.3 shows that there exists a feasible flow on this network and therefore the relaxed bound is, in this case, a feasible bound. Note that the bound on all of the other elimination sets is 24.

Example 3.5 (The First Phase)**.** *In Example 3.4, it is shown that there exists a solution to the problem with equality constraints for New York at a point bound of 28 and the set* $\{Montreal, Toronto, Detroit, Boston\}$ *is the only possible set with an optimal solution as the others had a point bound of 24. Since 28 is neither greater than the maximum possible points of New York or less than the current points of New York, no decision can be made and the second phase is needed.*

3.7 The Second Phase

In the second phase, only the elimination sets where teams can reach the point bound $\mathbf{p}$ are kept from the first phase and, for each set, there exists no scenario where every team in the set earns more points than the bound. The second phase is looking for sets where every team earns more points than the bound or if a team just reaches the bound, the tied team earns more wins. Observe that t_q has a certain number of earned wins, W_q,

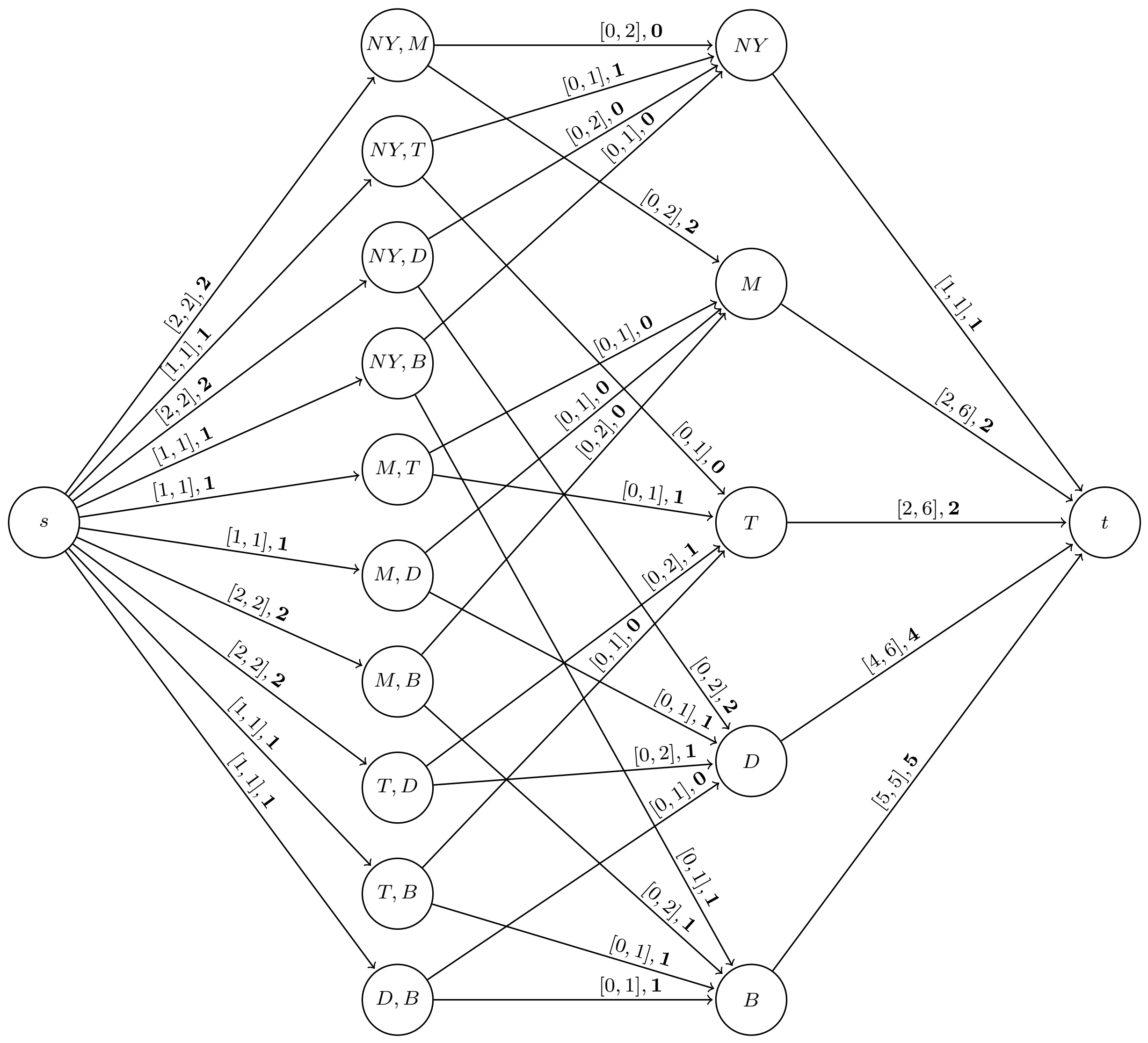


Figure 3.3: The feasible flow network from Example 3.4. Shows the elimination set of Montreal, Toronto, Detroit and Boston for New York with a point bound of 28. The arcs are notated with their upper and lower bound capacities. The bold number represents a possible feasible flow on the network.

initially and some maximum and minimum possible given $\mathbf{p}$. The range can be defined by observing the possible configurations of consolation points earned via overtime. The win range is defined more formally as,

$$W_q + \max\left((\mathbf{p} - P_q) - G_q, 0\right) \leq w_q \leq W_q + \left\lfloor \frac{(\mathbf{p} - P_i)}{2} \right\rfloor \quad . \tag{3.18}$$

Given the range of win values, each value is checked while bounding the wins of t_q, denoted $\mathbf{w}$. For each set, it is determined whether the teams can collectively exceed the point bound or reach the point bound and exceed the win bound, where the win bound is some possible win value of t_q. These problems are solved in a similar manner to the previous point bound calculation but with adjusted needs. There are four different cases that must be considered. First, some teams have have already exceed the point bound, therefore, their need is zero. Second, some teams may be able to reach the bound but would have more wins than the win bound. Therefore, some teams need to win at least as many games as needed to reach the point bound. Third, some teams may reach the bound but must require more wins to ensure that the wins are exceeded. Note that earning a single extra win is sufficient as a single extra win will give a team an additional point and thus not be tied on points. The last need that must be adjusted is that of t_q which must earn exactly $\mathbf{w}$ wins. By subtracting from the current value, W_q, t_q's need can be determined. The needs and conditions are formalized in the following equation.

$$n_i = \begin{cases} 0 & \text{if}\,(\mathbf{p} - P_i) - G_i < 0 \\ (\mathbf{p} - P_i) - G_i & \text{if}\,(\mathbf{p} - P_i) - G_i + W_i > \mathbf{w} \\ (\mathbf{p} - P_i) - G_i + 1 & \text{if}\,(\mathbf{p} - P_i) - G_i + W_i \leq \mathbf{w} \\ \mathbf{w} - W_i & \text{if}\,(p_i = \mathbf{p}) \wedge (w_i = \mathbf{w}) \end{cases} \quad . \tag{3.19}$$

Lemma 3.7 (Needs Satisfy the Constraints)**.** *A team t_i meeting the needs described in Equation 3.19 has more points or equal points and more wins if any of the first three conditions are true. If not they must be tied as defined by the fourth condition.*

Proof. The first condition states that if t_i lost every game but lost them in overtime then the t_i would have more points than the point bound. Under this condition, t_i could win none of their games and have more points. The second condition states that if by reaching the point bound, t_i has more wins then they only need to earn enough points to reach the point bound. The third condition states if t_i does not have more wins than $\mathbf{w}$ when reaching the point bound they must earn a single extra win. Observe that if t_i earned a single extra win beyond the minimum required to reach the point bound, while losing all of their other games in overtime, t_i would have more points. Therefore, it is sufficient for t_i to win only one more win than the minimum. The fourth criteria states that if t_i must be tied with the bounds at the end of the season then the number of points and wins should be equal to the point bound and win bound, respectively. Any other solution would have t_i fail to be better than t_q. □

Given the needs from Equation 3.19, this problem is formulated as a feasible flow problem as described in Section 3.6.2 except with new needs. If there exists a solution then t_q would need an extra point beyond $\mathbf{p}$ in order to guarantee a playoff spot. If $\mathbf{p}+1$ is greater than mpp_q then t_q cannot guarantee qualification. Otherwise, $\mathbf{p}+1$ points can be returned. If there are no solutions, then the needs must be checked for equality. This is to determine if any set could earn more points or equal points and equal or more wins. This can be done by modifying Equation 3.19 so that teams which need no extra wins to reach the point and win bound need only a minimum number of wins to reach the point bound as opposed to before where the team needed an extra point to exceed the win bound. If there is no solution under the modification for equality for any set, then there is no scenario where t_q does not qualify when earning $\mathbf{p}$ points and that value is returned. For any sets which have a solution, then the sets must be tested under the third phase.

Example 3.6 (The Second Phase)**.** *Referring to the example described in Section 3.3 and last expanded in Example 3.5, the tie-breaking methodology is expanded. First, it must be determined how many wins New York could earn given that $\mathbf{p} = 28$. Using Equation 3.18, New York must have between 11 and 13 wins if they earn 28 points. Each of these possible win values affects the needs of the teams. Using Equation 3.19, note that the first condition where teams could earn more than 28 points without winning a single game (i.e., only on overtime losses) does not hold for any of the teams as each team needs more than seven points to reach $\mathbf{p}$. For the second criterion, it must be determined how many wins a team absolutely must earn to reach the bound. That calculation of the condition is described below,*

$$\begin{aligned}
\textit{Montreal:} &\quad (28-18-7)+8 = 11,\\
\textit{Toronto:} &\quad (28-17-7)+7 = 11,\\
\textit{Detroit:} &\quad (28-16-7)+6 = 11,\\
\textit{Boston:} &\quad (28-14-7)+4 = 11.
\end{aligned}$$

The most important observation to make here is that none of the teams can possibly exceed even the lowest win value possible for New York without earning another win. This means that the third criteria from Equation 3.19 would have to be enforced for all teams and each team would have their need increase by one. Recall, however, from Example 3.5, that Boston can not increase their need as Boston requires all of their seven games just to reach 28 points. Therefore, one team cannot earn more than 28 points or exactly 28 points and more than 11 wins under any situation. Since Boston cannot earn more than 11 wins, win values of 12 and 13 can be pruned. The previous equation shows however that the teams could be tied with 28 points and 11 wins. Thus, the only remaining set has a point bound of 28, the only eligible win bound is 11 and the third phase is required to decide whether New York has clinched with 28 points.

3.8 The Third Phase

The third phase of tie-breaking is the most complicated of the tie-breaking constraints. It states that teams who are tied in terms of both points and wins are compared in terms of the number of points earned only against those teams who have also earned exactly that number of points and wins. The most difficult thing to determine about this constraint is that until the final points and wins are calculated for each team the exact composition of the set of tied teams is unknown. This problem is avoided by considering each set of possibly tied teams separately. Recall that the tie-breaking set, TB_q, is the set of teams tied with t_q such that, for every $t_i \in TB_i$, $p_i = \mathbf{p}$ and $w_i = \mathbf{w}$ where $p_q = \mathbf{p}$ and $w_q = \mathbf{w}$. Note that only the sets of teams that are tied with t_q are important as the positional arrangement of other tied teams is not relevant to solving the problem.

Many of these teams in the problem could not possibly be a part of a tie-breaking set. Given that the number of points and wins that a team must earn is fixed, only the number of points earned from overtime losses varies. As well, with larger sets, it is often not possible for all teams to be simultaneously tied.

To prune the possible membership of the tie-breaking set, a similar mechanism to the one used in the second phase is introduced. The major difference is that, unlike the second phase where the fourth criteria of Equation 3.19 only applied to t_q, here any team in TB_q will compute their need using this criteria. Any possible tie-breaking set where there is no feasible flow can be pruned. The equality modification to Equation 3.19 from 3.7 would also be applied.

Once the tie-breaking sets have been pruned, a feasible solution to Constraint 3.14 must be found. Unfortunately, this can no longer be represented in the flow network formulation used so far. At this point, a constraint programming solver is used to determine the feasibility of the final constraint. The basic model is extended to take advantage of the enumeration from the previous phases. First, the constraint that at least eight teams must be better is removed as this is ensured by enumerating in the first phase. Second, the complicated disjunction (Constraint 3.5) is replaced with a series of simpler constraints, described below, that can be expressed more easily and propagated earlier. Last, the optimization criterion can be removed as the tight bounding has turned the problem into a decision problem.

To reduce the burden of the disjunction, four different classes of teams are identified that exist in the problem: those teams who are in the elimination set and those that are not and, concurrently, those teams which are in the tie-breaking set and those who are not. The constraints that must be applied in each situation are different.

The first constraint that can be added is that teams which are members of the tie-breaking set must have exactly $\mathbf{p}$ points and $\mathbf{w}$ wins. Additionally, those tie-breaking set members who are also members of the elimination set must have more points against other tie-breaking members than t_q while those who are not members of the elimination set do not have to enforce this. If a team is neither a member of the elimination set nor the tie-breaking set then all constraints can be relaxed as those teams do not have to meet any bounds. Those teams who belong to the elimination set but not the tie-breaking set have to earn more points or equal points and more wins than t_q. However, as noted in the second phase, to earn enough points to exceed the bounds, it is only necessary to enforce that the teams reach their need values. By enforcing the needs, the teams necessarily satisfy at least one term of the disjunction as shown in Lemma 3.7.

Example 3.7 (The Third Phase)**.** *Referring to the example described in Section 3.3 and last expanded in Example 3.6, recall that there is a single eligible elimination set with* $\boldsymbol{p} = 28$ *and* $\boldsymbol{w} = 11$*. Note that since Chicago could only earn 24 points even if Chicago won all of their games, Chicago is not a candidate for the tie-breaking set. Now, the set of New York and Boston combined with any subset of the remaining teams is a valid tie-breaking set. However, recall from Example 3.6 that each team needs to earn at least one more win (and thus point) in order to not be tied. This would increase the need by at least one of either Montreal, Toronto or Detroit. Observe from Figure 3.3, that none of the teams exceeded their minimum capacity and no further points could be earned by any team. Therefore, we skip showing that the only valid tie-breaking set is the one that contains the entire set and New York as none of them could earn a single point without causing infeasibility for another team. Now all that remains to be shown is that the constraint program described by the elimination set, point bound, win bound and tie-breaking set has a solution where all of the teams in the elimination set has more points against teams that are tied than New York. In fact, there is no solution where this is true. New York can earn at most seven points against Chicago and Montreal must earn eight points against Chicago as there was no slack in the feasible flow to allow Montreal or New York to drop games from teams outside the set. Therefore, New York has 21 points against teams in the set and Montreal only has 20 points against teams within the set and, therefore, New York needs 28 points to guarantee a playoff position.*

3.8.1 Symmetry Breaking and Redundant Constraints

While this model is a correct model, additional symmetry breaking and redundant constraints are added to ensure that solutions are found quickly. It is a common modelling technique to add additional constraints, either statically or dynamically, to remove symmetries and dominances from the solution set. This technique is combined with the initial enumeration to remove symmetries and dominances that could not be easily detected without the enumeration.

The most obvious dominance constraint is that elimination sets which do not have an upper bound equal to the point bound can be discarded.

There is little true symmetry in the NHL Playoff Qualification problem as each team often has a different number of games remaining, points, wins and different opponents for their remaining games. However, by using the phased approach combined with enumeration, it is possible to identify classes of teams where symmetries can be applied to reduce the search space of the problem. Recall from above that the classes are based on the team's membership in either the elimination or tie-breaking set or both.

The most obvious of the symmetries is that teams that belong to neither set are unconstrained under the specific set restriction being applied. If these teams play games between each other, the result is not affected regardless of the outcome of the game. Therefore, values can be arbitrarily assigned to the win variables for games between teams in that class. As well, if teams in this class earn overtime points it does not affect any constraint and the overtime variables for each team, therefore, can be fixed to zero.

Another symmetry is that teams that are members of the elimination set but not the tie-breaking set win all of their games against those who belong to neither including those in the opposite conference. Unlike those teams who belong to the tie-breaking set, neither their wins nor their points are constrained by an upper bound. Since none of the teams whom they are defeating need the points, if there is a solution then there is also a solution with those wins.

For those teams in the tie break, it is possible to add a redundant constraint which also constrains the number of overtime losses.

There is a useful relationship between teams within the tie-breaking set and overtime losses. There are three groups of teams in the tie-breaking set: those teams that are also in the elimination set that are trying to earn as many points as possible to be better than t_q, those teams that are not in the elimination set which just need to reach the bounds and t_q. It is often possible when a team from within the tie-breaking set loses a game to set the overtime loss value immediately. For the first group, any time they lose games against teams also in the tie-break, it is possible to immediately assign the corresponding overtime loss values to as high as possible without violating the redundant constraint on the number of overtime losses allowed. For those teams that are not in the elimination set and are not t_q, any time the lose games regardless of whom they are playing, as many overtime points as possible without violating the overtime loss constraint should be assigned. Every time t_q loses a game against a team that is not in the tie-break, the overtime loss should be assigned as long as the overtime loss constraint is not violated.

Once the win values for every game that involves the tie-breaking set has been determined, the remaining values can be propagated without search. First, the remaining win variables for the teams in the elimination set but not the tie-break set must be assigned.

However, this problem is identical to the problem solved in the second phase and can be solved in polynomial time. The only variables that would be left to be set are the overtime loss variables for t_q against teams within the tie-breaking set and those by the elimination set members also in the tie-breaking set. However, at this point it is possible to determine, the number of overtime losses earned by each team and assign them based on a predefined order which ensures a solution if possible.

3.8.2 Pruning Values from Constrained Teams via Flow Manipulation

The feasibility of the tie-breaking set depends on whether there exists a feasible flow equal to the needs of the teams in the flow network of the type represented by Figure 3.4. An important observation that can be made is that any feasible flow is a valid assignment of the win variables of the teams in the elimination and tie break set. The variables within the solver can be pruned by modifying an already existing flow to contain a specific test value using a method adapted from Maher et al. [40]. If there is a flow that contains the value then there is a support for that value and that value is kept. If not, the value is pruned from the domain of the variable. The idea is similar to the idea used in the Ford-Fulkerson algorithm. However, in this case, a path between two internal nodes is desired.

To reduce the practical complexity of the algorithm, the residual graph is reduced to only those components that will be updated. In a graph containing a feasible flow, the edges out of v and into w are completely saturated. Since any modification must also be a feasible flow, these edges must remain saturated and any modification should not alter these edges. The other reduction that can be made to the graph depends on the symmetry between nodes representing teams and the links to their matched games. This allows us to remove the nodes representing the matched games and link the nodes directly together.

Example 3.8. *Examine Figure 3.4b and note that in the residual graph links into v and out of w are saturated and can be removed. Also observe that the edge from node 1 to node $(1,2)$ is the same as the edge from node $(1,2)$ to node 2. Therefore, we can remove node $(1,2)$ and directly link $(2,1)$. Figure 3.5 shows the reduced pruning graph along with a single variable update.*

3.9 A Note About Division Leaders

An intentional omission from the preceding discussion of the solution is the issue of division leaders and their special status in sports leagues. A division leader is guaranteed a playoff spot regardless of how it would have actually ranked had it not been at the top of its

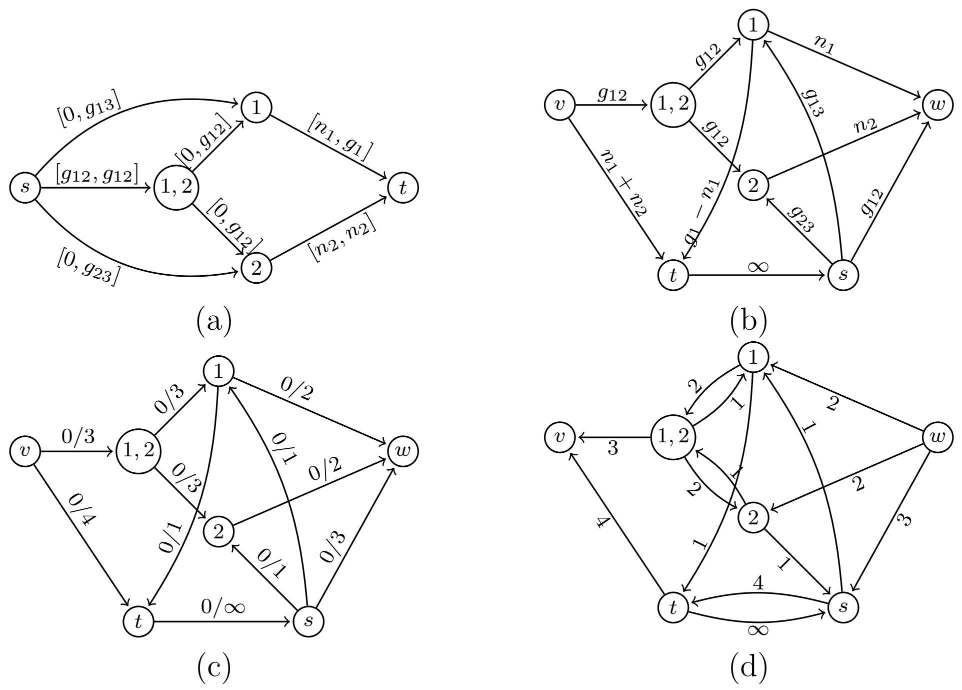


Figure 3.4: **(a)** A network flow with three teams. Team 1 has a lower bound constraint on the number of wins and is in the elimination set and not in the tie break set, team 2 is in the tie break set and has a fixed number of possible wins, and team 3 is in neither the elimination set or the tie break set and has no bounds on either points or wins. **(b)** A flow graph transformed to remove the lower bound capacities. Two additional nodes are added v and w. A feasible flow exists in the original graph if the maximum flow is equal to the sum of the lower bounds on the original graph $(n_1 + n_2 + g_{12})$. **(c)** An example flow graph for three teams where Team 1 must earn between 2 and 3 games, Team 2 must earn exactly 2 games and Team 3 is unbounded. **(d)** The residual graph containing a maximum flow.

division. The complication of these division leaders arises from the tie-breaking constraints used. First, it is shown that at most one division leader could be weaker than t_q.

Lemma 3.8 (At Most One Weak Division Leader)**.** *At most one of the three division leaders can have fewer points or equal points and fewer wins than t_q if t_q does not clinch a playoff spot.*

Proof. It is impossible for the leader of t_q's division to have fewer points or equal points and fewer wins as the team could not be a division leader as t_q would be ranked ahead of the team within that division. So at least one team in t_q's division must have at least

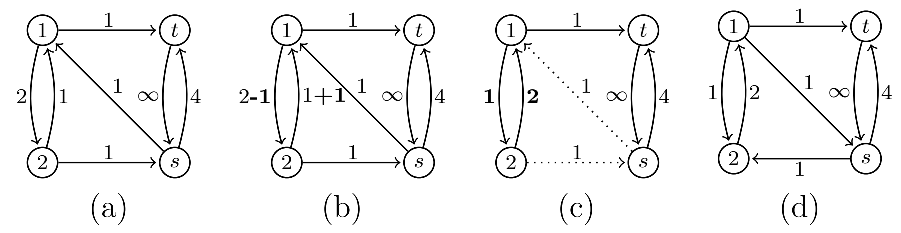


Figure 3.5: Reduced Pruning Graph. **(a)** shows the reduced residual graph of Figure 3.4. In **(b)**, the link between nodes 1 and 2 is reduced and the link between nodes 2 and 1 is increased, which ensures the constraint that the flow between them equals some mutual capacity. **(c)** shows the path that is found from node 2 to node 1 correcting the imbalance. Once a path is found, the flow is redirected and the opposite edges are updated by the change. **(d)** shows the new stable solution showing support for the assignments of $w_{12} = 1$ and $w_{21} = 2$.

equal points and equal wins. Suppose, however, that the two other division leaders have fewer points or equal points and fewer wins. Note however this also means that every other team in both divisions will have fewer points or equal points and fewer wins than t_q. Since at most four other teams from t_q's division plus the two division leaders would qualify, t_q must qualify in seventh at worst. Therefore, at most one division leader can have fewer points or equal points and fewer wins if t_q does not qualify for the playoffs. □

This can be effectively dealt with by simply tweaking how elimination sets are generated. If a team is a division leader that has fewer points or equal points and fewer wins then there are obviously no other teams that could be in the elimination set as they would fail the constraints. Therefore, any elimination set that contains only a single team from a division that does not include t_q can simply drop that team. If the set contains no members from a division, again not containing t_q, eight sets of seven are generated by selectively leaving one team out of the original eight team set.

The more complicated case is when every member of the elimination set from a particular division is also in the tie-breaking set. In this case, it is possible for a team t_i to fail to be better than t_q but to still also be the division leader. The reason for the discrepancy is that when t_i is compared using the third tie-breaking criteria there are two different sets of teams: the set of all teams which are tied and the set of all teams from the same division that are tied. It is possible for t_i to lose the tie break in the former and win in the latter which still assures them a playoff spot as division leader. These situations are very rare and can be handled individually when the situation arises. First, if this pertains to a division not containing t_q, the problem is solved normally and, if a solution is returned, a valid solution is obtained. If no solution is found or if t_q is possibly a division leader, further steps must be taken. Given there is only a small number of possible division leaders, the problem is solved enforcing a different division leader each time. The enforced division leader does not need to have the tie-breaking constraint enforced for the overall solution but it does have to be enforced against its divisional rivals.

3.10 The Elimination Problem

The focus of the explanation has been on the solver for the qualification problem but the basic mechanics are applied in the same manner for the NHL playoff elimination problem. The major difference is how the point bound is calculated given a qualification set. Instead of finding the maximum point value of the weakest team in the set, the minimum possible point value of the strongest team is found using the feasible flow algorithm. Once the point bound is established, the other major difference is that if a solution is found then the point bound is returned and if there is no solution then the point bound plus one is returned.

Another difference with elimination problems is with division leaders. It must be checked first if t_q could be a division leader for less points. Note this can be integrated seamlessly with the normal procedure by adding the set containing only the other division members. If this set has a better point value, it is kept and, if not, it is pruned like other sets.

3.11 Experimental Results

The solver was implemented in C++ using the Boost Graph Library [64] for the feasible flow calculations and ILOG Solver [31] to solve the final constraint model. The calculated results are compared against those shown in the Globe and Mail [26] for the 2005-06 and 2006-07 season. The 2006-07 season results are used to calculate the minimum points needed to clinch a playoff spot. In total, determining the bound for all 30 teams on all 181 game days of the 2006-07 NHL season (5430 problems) took a little over 46 hours on a Pentium 4 PC. Each individual instance, representing a team at a given date, took less than ten minutes to calculate the bound and those problems near the end of the season, where the results matter the most, were calculated in seconds. The dates at which various teams would have clinched using the various scoring models used and proposed by the NHL was compared.

To generate the qualification problem instances, the results from the 2005-06 and 2006-07 seasons were obtained and broken into separate days. To create instances from the results for each day of the season, the number of points and wins earned up to that point in the season was calculated.

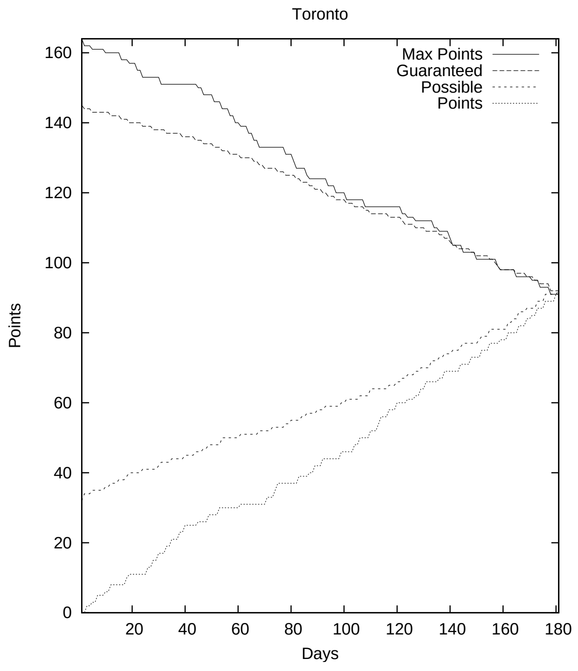
Each of the problem instances for the 2005-06 and 2006-07 seasons was tested and the date of qualification for each of the teams was determined. For most instances, the solving time was a fraction of a second. When comparing the results to the results posted in the Globe and Mail, the exact results generated show qualification earlier for nine teams during the 2005-06 season and for four teams during the 2006-07 season. Interestingly, the Globe and Mail never announced clinching before the actual date in 2005-06 and 2006-07. The results of this experiment are shown in Table 3.4. In this table, entries with multiple dates are due to the absence of Sunday editions or unreported results where it was unclear on which day the result would have been reported.

Qualification was shown earlier than the Globe and Mail by as much as five and four days for the 2005-06 and 2006-07 seasons, respectively. Note that the largest difference is for the first team to qualify in both seasons. It is possible that this is due to the paper not calculating the results because they did not realize that teams had clinched playoff spots. Even disregarding these results, discrepancies as large as four days and two days were found for the 2005-06 and 2006-07 seasons, respectively.

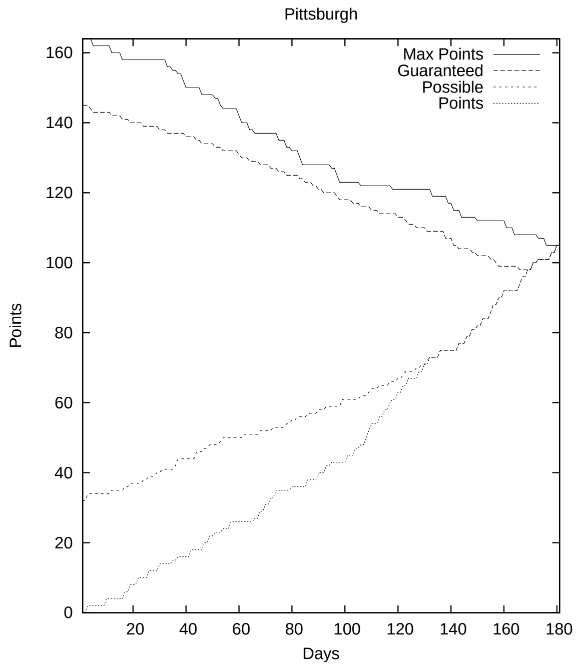
Table 3.4: The results of qualification compared against the published Globe and Mail results. Only results that differ are shown.

2005-06			2006-07		
Team	Optimal	Globe and Mail	Team	Optimal	Globe and Mail
Ottawa	Mar 22	Mar 27	Buffalo	Mar 18	Mar 22
Montreal	Apr 14	Apr 18	Atlanta	Apr 2	Apr 4
Buffalo	Apr 4	Apr 5	Detroit	Mar 24	Mar 25 or 26
New Jersey	Apr 12	Apr 14	Nashville	Mar 23	Mar 24
Calgary	Apr 8	Apr 9 or 10			
Colorado	Apr 13	Apr 15			
San Jose	Apr 13	Apr 14			
Dallas	Mar 31	Apr 1, 2 or 3			
Nashville	Apr 9	Apr 10			

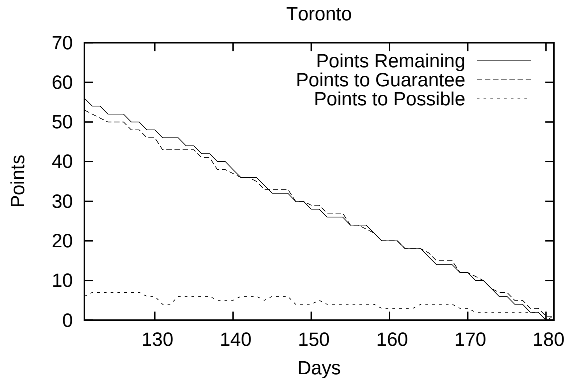
The bound result is plotted against both the current points of the team and maximum possible points of the team. If the bound result is greater than the maximum possible points, then the team is no longer able to guarantee a playoff spot. If the current number of points is equal to the number of points needed by the team then that team has clinched a playoff spot. Figure 3.6 shows the result calculated for Toronto and Pittsburgh. Note that Toronto did not make the playoffs because the current points never reached the bound value. Also note that Toronto placed themselves in a position where guaranteeing a playoff spot was not possible and got lucky four times. In other words, they lost a "must win"



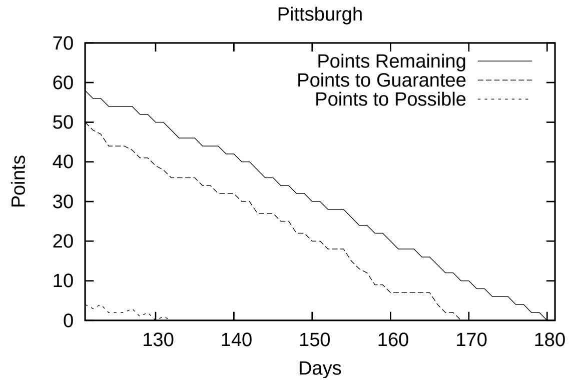
(a)



(b)



(c)



(d)

Figure 3.6: **(a)** and **(b)** The minimum number of points needed by Toronto and Pittsburgh to guarantee or possibly qualify for a playoff spot in the 2006-07 NHL season. **(c)** and **(d)** The difference between current points by Toronto and Pittsburgh and the maximum points, number of points to guarantee and number of points to possibly qualify in the 2006-07 NHL season.

game five times during the 2006-07 season while Pittsburgh was never in that situation. Another interesting feature is that, in both graphs, the bound on points, 145, at the start of the season to guarantee a playoff spot was quite high.

Table 3.5: Shows some of the features that can be highlighted by calculating the minimum number of points needed to guarantee a playoff spot.

Feature	Value	Team(s)
Earliest day where a team could not guarantee	64 days	St. Louis
Most days where a team could not guarantee	118 days	St. Louis
Most times a team got lucky	4	Toronto, Boston and NY Rangers
Number of teams that got lucky and earned a spot	2	NY Islanders and NY Rangers
Number of teams that got lucky but failed to earn a spot	7	Toronto, Boston, Washington, Carolina, Edmonton, Phoenix and Columbus

Table 3.5 shows an overview of the results of the 2006-07 NHL season in terms of the minimum points needed to guarantee a playoff spot. One interesting observation that can be made from this table is that of the nine teams that got a second chance only two of those teams ended up earning a playoff spot. As well, of those seven teams, two of them had four chances to make the playoffs after losing a must win game. Another interesting note is that St. Louis could not guarantee a playoff spot after only the sixty-fourth game day and never recovered during the final one hundred and eighteen game days.

Once the solver has been designed for one scoring model, it is relatively straight-forward to translate the model to other scoring systems. The instance generator was modified to calculate the points correctly for each of the different scoring models and the instances were run for all of the scoring models used by the NHL as well as a proposed model for the NHL which was used in the 2010 Winter Olympics. The results of this experiment can be found in Table 3.6. The average of number of days remaining when teams clinch in a given season can vary by more than two days depending on the scoring model used.

The most interesting effect of changing the scoring model was to observe which teams qualified for the playoffs. In the case of the 2006-07 season, the Toronto Maple Leafs would have qualified for the playoffs under any other scoring model than the one used that season all other things being equal (see Table 3.7). Another interesting note is that Edmonton, who were the Western Conference Champions in 2005-06, would not have made the playoffs if the proposed scoring model had been imposed and their spot would have gone to divisional rival Vancouver (see Table 3.8). It should be noted that these results assume that the games would end in the same way regardless of the scoring system. However, research in economics has shown that teams modify the way they play depending on the

Table 3.6: Average days remaining by scoring system for the 05-06 and 06-07 seasons.

	Average Days Remaining	
Scoring Model	2005-06	2006-07
Historic	12.75	11.75
Overtime	11.56	12.81
Extra Point	11.75	10.63
Current	11.94	10.75
Proposed	13.13	10.13

scoring model [9]. Even given this fact, it is interesting to note that some team strategies would have been good enough to secure a playoff spot under any scoring model while other teams strategies only ensure qualification under the current scoring model, for example, Tampa Bay in 2006-07.

Table 3.7: Date of clinching under different scoring models in the Eastern conference of the NHL in 2006-07.

Team	Historic	Overtime	Extra Point	Current	Proposed
East					
Toronto	Apr 8	Apr 8	Apr 8	—	Apr 9
Ottawa	Mar 19	Mar 18	Mar 21	Mar 25	Mar 21
Montreal	—	Apr 8	—	—	Apr 7
Buffalo	Mar 24	Mar 21	Mar 23	Mar 18	Mar 22
Boston	—	—	—	—	—
NY Islanders	Apr 8	—	—	Apr 9	—
NY Rangers	—	—	Apr 9	Apr 6	Apr 7
...	...	...	...	...	...
Tampa Bay	—	—	—	Apr 6	—
Florida	Apr 6	—	—	—	—
Atlanta	Apr 7	Apr 1	Apr 2	Apr 2	Apr 7
Carolina	—	Apr 2	Apr 6	—	—

The combination of the various techniques generates a set of models for each problem that can be easily solved. Each component builds on the techniques used previously. The initial enumeration is used to deal with the implication constraints and the cardinality constraint on the indicator variables. Once the sets have been enumerated, it is easy to prune the sets since bounds can be calculated efficiently. The phased solver method

Table 3.8: Date of clinching under different scoring models in the Western conference of the NHL in 2005-06.

Team	Historic	Overtime	Extra Point	Current	Proposed
Vancouver	—	—	—	—	Apr 14
Edmonton	Apr 14	Apr 14	Apr 18	Apr 14	—
Calgary	Apr 6	Apr 6	Apr 6	Apr 8	Apr 6
...	...	...	...	...	...

takes the idea of enumeration and extended that to disjunctions as well as implication constraints. Lastly, once the enumeration has fixed both the points and wins earned by the distinguished team, it is possible to fix many of the other variables due to symmetry.

Table 3.9: The counts of problems in the 2007-08 season solved via the various stages of the solver. Positively solved instances means a solution was found and the bound must be increased. Negatively solved instances means that bound was valid for that instance. Any problem without a definitive solution was passed to the next phase of the solver.

Solver Stage & Result	Number of Instances (/5430)	Cumulative Percentage
First Phase	1212	22%
Second Phase (Positively)	2249	
Second Phase	1524	92%
Third Phase (Positively)	338	
Third Phase (Negatively)	107	100%

Table 3.9 shows the results breakdown of the solver in terms of its phases. The first phase solves 1212 of the 5430 of the problems and in the second phase, with tie-breaking on wins, a further 3773 problems are solved, which makes up about 92% of the problems. However, the remaining 8% of problems require a backtracking constraint solver to calculate the final result in the third phase. Also, note that in 47% of the total instances, the answer differs from the initial lower bound.

Table 3.10 shows the improvements of the various techniques. Implementing a sum of variables indexed by a set of indices from a set variable proved infeasible using ILOG Solver 4.2 [31], so the simple model was not implemented in its pure form. From the results, implementing enumeration allows some problems to be solved but it must be combined with other techniques. The largest improvement was from using enumeration and decomposition

Table 3.10: Results from the 2006-07 season. Instances are solved for every game day, working in reverse order from the end of the season (Apr 8) to the beginning of the season (Oct 4). There are a total of 5430 instances.

	Enumeration	Bounding	Phased Solver	Symmetry Breaking
Timeouts (300 s)	5075[1]	4217	2	0
Latest Date of Failure	Apr 4	Apr 4	Oct 5	N/A
Earliest Problem Solved	Mar 18	Dec 5	Oct 4	Oct 4

[1] Due to time constraints, timeouts for Enumeration are estimated. Once a team timed out ten times, it was extrapolated that all of the remaining instances would time out. This is a relatively safe assumption since the instances get larger towards the beginning of the season and the gap between the bound and the current points increases.

to create a phased approach. Symmetry breaking constraints were added to solve the last remaining problems.

3.12 Summary

As the season winds down, the fans of the NHL are interested in knowing how far their team is from clinching a playoff spot. A method for calculating the minimum number of points that must be earned in order to ensure that the team reaches a playoff spot was presented. This calculation is efficiently computed by using a multi-stage solver that combines enumeration, flow network calculations and backtracking search.

This work represents the first complete and efficient solution to the NHL qualification and elimination problems. The key to scaling up the constraint programming approach was a combination of enumeration and additional symmetry breaking and redundant constraints. By introducing symmetry and redundant constraints to the model, the amount of search necessary to find a solution or to confirm that no solution was possible was reduced. As well, the costly work associated with the calculation of the actual division leaders was avoided as much as possible.

The 2005-06 and 2006-07 seasons were used to verify that the solver could solve realistic problems. Solving an individual instance only took a fraction of a second and all instances of the 2005-06 and 2006-07 seasons could be solved in a several minutes while respecting all tie-breaking rules. As well, clinching and elimination results could be announced as much as five days earlier than the Globe and Mail [26]. Observing the effect of scoring model on playoff qualification, it was found that qualification under scoring models could vary by by more than two days and changing the scoring model caused different teams to qualify for the playoffs.

A side effect of calculating the number of points need to qualify for the playoffs is the ability to determine when the team is in danger of losing control of its destiny. These games, often described by coaches as “must win” games, can be identified as the loss reduces the maximum possible points to below the bound of the team. Nine different teams in the 2006-07 NHL season were identified that lost control of their fate and then gained that control back through mistakes by their opponents. Only two of these teams took full advantage of this situation and clinched a playoff spot. Of the nine teams which experience this event, three of them experienced it four times.

In the next chapter, a series of different manipulation strategies is examined a from computational perspective. Work from social choice theory and winner determination problems is adapted to create algorithms for manipulations in cup competitions and round robins.

Chapter 4

The Manipulation of Sporting Tournaments

The Gibbard-Satterthwaite theorem states that, unless dictatorial or unfair, voting systems are always manipulable. One possible escape proposed by Bartholdi, Tovey and Trick is that the manipulation may be computationally too difficult to find [10] (but see [68] for discussion about whether manipulation is hard not just in the worst case). Like elections, sporting competitions can also be manipulated. For example a coalition of teams might throw games strategically to ensure that a desired team wins or a certain team loses. Another example of manipulation would be to organize the teams in a cup competition so that the desired match-ups occur.

Manipulating a sporting competition is slightly different from manipulating an election as in a sporting competition the voters are also the candidates. This means that there are no votes to consider as only the two teams participating in a match can change the outcome of the game.

In this chapter, manipulation is viewed from two different computational perspectives[1]. First, viewed from the perspective of theoretical worst-case complexity, the theory of computational complexity used in the election literature, specifically the work on sequential majority voting and Copeland voting, is applied to common sports competitions. As well, several different variants are discussed including how minimal sized coalitions could be constructed for various sports competitions. Second, viewed from the perspective of practical complexity, it is observed how difficult problems are in practice. It is shown that realistic problems can be solved extremely quickly using, in this case, constraint programming.

Two common types of competitions are discussed in this chapter: cup competitions

[1]Portions of this work have previously been published in [60]. This work was co-authored with Toby Walsh.

and round robin competitions. These two types of competitions and variants of these competitions cover most if not all sports competitions. Cup competitions pit pairs of teams against each other until only a single team remains. Round robin competitions allow each team to play every other team and the total of the results is summed at the end. The team with the most points is declared the winner.

Three types of manipulations are discussed in this chapter: constructive manipulation, destructive manipulation and seeding manipulation. A constructive (or destructive) manipulation attempts to modify the outcome of games so that a desired team wins (or loses). A seeding manipulation attempts to determine a seeding of a cup competition so that a desired team wins the competition. It is assumed that there is a tournament graph that describes the expected outcome of all fair games between opponents. Manipulating a game in constructive or destructive manipulation modifies the tournament graph directly. Since it is hard for a team to play better than it can, we consider manipulations where teams in the coalition are only able to throw games. By comparison, in an election, voters in the manipulating coalition can mis-report their preferences in any way they choose.

I first address manipulating cup competitions, also known as single elimination competitions. It is shown that a coalition can determine a manipulation strategy in polynomial time. This technique is extended to look at a minimization variant where a minimal set of conspirators are found among a set of willing cheaters. Variations of the basic tournament structure like double elimination tournaments and reseeding cups are also discussed.

I show that round robins, using the most common scoring model (a win or a loss), can be manipulated in polynomial time by a coalition of cheaters. As well, it is determined which of the remaining set of scoring models can be manipulated in polynomial time and for which it is NP-Hard to do so. When extending the work to qualification and elimination as discussed in Chapter 3 and the work of Ribeiro and Urrutia [54], the problem is almost always NP-Hard. However, the work in Chapter 3 and Ribeiro and Urrutia [54] shows that the problem of qualification and elimination for NHL hockey and Brazillian soccer can be solved in practice.

When discussing cups, there is another obvious manner in which they can be manipulated. If the manipulators could influence the construction of the schedule then the manipulator could change who wins the competition. The worst-case complexity of manipulating the seeding, the schedule of games, is unknown and remains an open problem [35, 28, 67, 70]. However, as shown later in the chapter, regardless of the complexity, it can be easy to determine seedings where the desired team wins given a random tournament graph. Even tightening the restrictions to which a seeding must conform does not necessarily make the problem hard.

Last, the combination of the various manipulations is discussed. First, the combination of cup manipulation and seeding manipulation is discussed and its practical complexity is analyzed. Second, the complexity of a multi-stage competition which has a round robin first stage and a final stage is derived. In many sporting tournaments, including the World Cup, this is the common competition structure. It is shown that if only a single team advances from each group then a manipulation strategy can be determined in polynomial time. If not, there are several variants where two teams advance from the group stage to the cup competition where a manipulation strategy can be determined in polynomial time. In general, determining if k teams advance from the group is NP-Complete [45].

At first glance, the notion of determining algorithms for manipulating tournaments has a particular negative social connotation. In some sense, this work could be used to empower the cheaters. An analogy can be made, however, to computer security where the security holes exist and failure to acknowledge their existence does not protect the tournaments from the abuse. As well, the assumption that the work generated here could not possibly have been discovered, in part or in whole, by the cheaters is unlikely to be true. As such, there seems to be some benefit to revealing the manipulation strategies so that the tournament organizers are aware of them and can attempt to combat them.

Before discussing the algorithms and results, some related work is presented, primarily in voting mechanisms (Section 4.1). Manipulations by coalitions on cup competitions are described in Section 4.2. In Section 4.3, it is shown that coalitions can manipulate round robin competitions. A different type of manipulation, one of organization, is discussed in Section 4.4 where the effect of different seedings of tournaments on the winners of tournaments is examined. Combinations of these manipulations are discussed in Section 4.5. Finally, some concluding remarks are made in Section 4.6.

4.1 Related Work

Tang, Shoham and Lin [66] address the direct manipulation of tournament graphs in team competitions by providing conditions for truthful reporting of player strengths. Tang et al. [66]'s method tries to encourage teams to rank their players honestly so that, when the teams compete in bouts, the best player on one team plays the best on the other, the second best plays the opposing second and so forth. An example of a competition where opponent matching is used is Davis Cup Tennis. The work presented in this chapter is similar in theme but uses different techniques.

Conitzer, Sandholm and Lang [18] give an algorithm to determine if a coalition can manipulate a cup-based election. In this chapter, their algorithm is modified to manipulate directly the tournament graph instead of the votes. Bartholdi, Tovey and Trick [10] discuss direct manipulations of the tournament under second-order Copeland, a round robin like rule with secondary tie breaking. Bartholdi et al. [10] use a single scoring model and results in this chapter extend the idea to all possible scoring models. Using the work of

Kern and Paulusma [34], the relationship between the manipulation of round robin competitions and winner determination in sports problems is established in Section 4.3. Altman, Procaccia and Tenneholtz [3] construct a social choice rule that is monotonic, pairwise non-manipulable and non-imposing. Round robin and cup competitions are monotonic as a single team losing a game does no better. Pairwise non-manipulability means that no two teams are better off by manipulating the tournament. I show that round robin and cup competitions are pairwise manipulable and that manipulations can be calculated in polynomial time.

The algorithms used to determine manipulations of the tournament can be modified to calculate the smallest number of manipulations needed. In this chapter, dynamic programming is added to Conitzer, Sandholm and Lang's algorithm to calculate minimal cup manipulations. Vu, Altman and Shoham [67] use a similar method to calculate the probability that a team wins the competition, which is a related but different problem. For round robin competitions, I show that there is a variation that adds weights to the flow network used to solve winner determination problems and uses a minimum cost feasible flow cost algorithm to determine the minimal number of manipulations. This allows minimal manipulations to be calculated in polynomial time for cup competitions.

Vu et al. [67] provide several results on determining probabilities of teams winning, given a seeding of the tournament. Hazon et al. [28] show that it is NP-Complete to determine whether there is a seeding where a team wins a cup with a given probability. This is similar to determining a possible winner given random reseeding except edges in the tournament are labelled with probabilities.

Lang et al. [35], Vu et al. [67], Hazon et al. [28] and Williams [70] have all discussed the probability of determining a seeding in the deterministic case where a given team wins. As yet, the complexity of determining a seeding when the results of the games are deterministic—a tournament graph is used to determine results—remains open. However, Lang et al. [35] show that determining the solution for the weighted case is NP-Complete. Both Hazon et al. [28] and Vu et al. [67] show that the problem of determining a seeding for a balanced tournament which maximizes the probability of a given team winning is NP-Complete. Williams [70] shows that certain classes of these problems can be solved in polynomial time. As well, she showed that there is high likelihood that random problems will be solvable in polynomial time. I tackle the open problem from a different perspective. Instead of determining the worst-case complexity, an experimental study is performed to analyze the practical hardness of problems. Further realistic restrictions are placed on the problems that have not been discussed in previous work.

4.2 Cup Competitions

Cup competitions are an interesting case because in order for a coalition to manipulate games they must detrimentally affect their own performance by throwing a game. This would suggest that the incentive for throwing the game would have to be strong. An example where this occurred is during the 1919 World Series where the White Sox deliberately threw games in the final and, in the end, gave away the championship. The reason for this was stated to be underpaid players being bought off by gangsters [6]. There are other motivations that are plausible. A coalition might employ destructive manipulation to ensure that a certain team loses, which they collectively do not want to win either for spite, jealousy or rivalry. Also, there is often some monetary reward to finishing in first place. A group of teams might form to promote a team because if they manipulate the tournament and split the winnings, this might increase their expected payout. An interesting question that is not addressed here would be whether it is possible to generate an incentive compatible payout mechanism which satisfies the property that every team is expected to earn more money by playing each game fairly but there still remains some incentive to win the tournament. However in real world circumstances, this is further complicated by the presence of outside money such as in the 1919 Black Sox scandal.

Cup manipulation strategies in the direct tournament manipulation case amount to determining for each coalition member the round in which they would throw a game. Background material pertaining to this work can be found in Section 2.5. Section 4.2.1 discusses how coalitions can be made minimal in the sense that each member must manipulate a game in order to achieve the goal and no unnecessary coalition members exist which did not need to manipulate. Manipulation of double elimination cups and reseeding cups is tackled in Sections 4.2.2 and 4.2.3, respectively.

The primary result of this work is that finding a constructive or destructive manipulation of the competition is polynomial. This is shown using the results in [18] which shows that a manipulation of an election using the cup rule can be found in $O(m^3n)$ time where m is the number of candidates and n is the number of voters. The basic algorithm (see Algorithm 4.1) is designed for fixed cups where the seeding at each round is predetermined.

The basic Conitzer-Sandholm-Lang (CSL) algorithm is a recursive method that treats each node in the cup tree (which is not a leaf) as a sub-election (see Algorithm 4.1). Conitzer et al. [18] note that a team wins a sub-election if and only if they win one of its children and they can defeat one of the potential winners on the other side. It is perhaps simpler to understand this algorithm from a bottom up perspective in terms of sports. Observe that if there are leaf nodes t_i and t_j and there exists an arc in the tournament (t_i, t_j) then t_i will win the match between t_i and t_j and, thus trivially, is a potential winner of the match between t_i and t_j. Now suppose that t_i is a member of the coalition. If t_i throws the game, t_j is also a potential winner of the match between t_i and t_j. Assume that

```
Algorithm:CSL($t_w$, $C$, $G$, $S$)
input  : A team $t_w$, a cup tree $C$, a tournament graph $G = (T, E)$, and a coalition
         of teams $S$
output: Returns true if $t_w$ can win via manipulation and false otherwise
winners ← PossibleWinners($C$, $G$, $S$);
if $t_w$ ∈ winners then
    return true;
else
    return false;
```

Algorithm 4.1: The pseudo-code for the main function of the CSL algorithm

there is some match in the middle of the competition with two sets of potential winners A and B. Any team from A is a potential winner of the match if there exists a team in B that they can defeat or if a coalition member in B throws a game. The same is true for teams in B. Therefore, there is a constructive manipulation if the desired winner is a member of the potential winners at the top node in the cup tree.

Theorem 4.1 (Constructive Manipulation of Cups). *Determining if a cup competition can be constructively manipulated using manipulations of the tournament takes polynomial time.*

Proof. The CSL algorithm examines at most $O(m^2)$ pairs of opponents as no two teams are compared more than once. Note that the original analysis provided a looser $O(m^3)$ bound on the number of comparisons, but this can be tightened by an observation of Vu et al. [67]. The difference between direct manipulation of the tournament and the method by Conitzer, Sandholm and Lang is that determining if a team could defeat another team meant summing all values of the n voters requiring $O(n)$ time whilst in the direct manipulation of the tournament this can be done in constant time. Therefore, constructive manipulation of the tournament in a cup competition takes just $O(m^2)$ time. □

Example 4.1. *To illustrate how manipulations occur in cups, assume there is a sixteen team example with coalition members t_3, t_5, t_{11}, t_{12}, t_{13} and t_{15}, who wish to manipulate the tournament so t_7 wins. Also given are the tournament graph in Table 4.1 and the cup in Figure 4.1, which shows the seeding of teams. t_1 is the expected winner of the cup if no manipulation occurs. Applying Algorithm 4.1, the potential winners are calculated at the first stage of the competition. If there is a coalition member in a pair of teams then both teams proceed otherwise only the expected winner makes it to the next round. At the next round, teams may potentially face more than one opponent depending on which manipulation occurs. If a team can beat any of their possible opponents or if one of the*

```
Procedure: PossibleWinners($C$, $T$, $S$)
input  : A cup tree $C$, a tournament graph $G = (T, E)$ and a coalition of teams $S$
output : Returns the set of possible winners of the cup tree via manipulation of the
         tournament by the coalition
if leaf($C$) then
    return $\{C\}$;
else
    winners $\leftarrow \{\}$;
    LeftWinners $\leftarrow$ PossibleWinners(left($C$), $T$, $S$);
    RightWinners $\leftarrow$ PossibleWinners(right($C$), $T$, $S$);
    forall $t_i \in$ LeftWinners do
        if $\exists t_j \in$ RightWinners such that $(t_i, t_j) \in E \vee t_j \in S$ then
            add(winners, $t_i$);
    forall $t_j \in$ RightWinners do
        if $\exists t_i \in$ LeftWinners such that $(t_j, t_i) \in E \vee t_i \in S$ then
            add(winners, $t_j$);
    return winners;
```

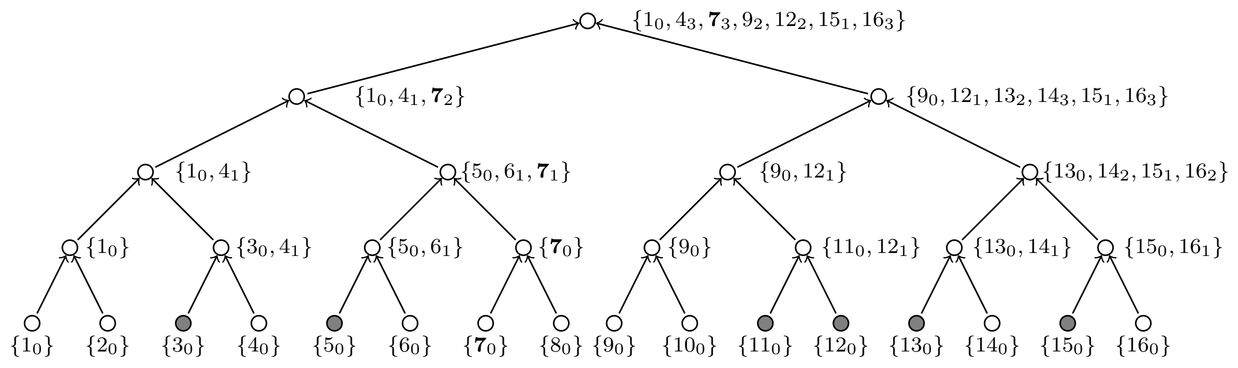
possible opponents is a coalition member who could lose to them by manipulation, then we add that team as a possible winner of that round. For example, t_7 loses to both t_5 and t_6 in a fair match but since t_5 is a coalition member they could manipulate that game and t_7 continues to the next round. Continuing this reasoning, the possible winners are generated at each level. Since t_7 is a possible winner on the final level, they can be made a winner of the competition by manipulation.

Observe that destructive manipulation of a competition using tournament manipulations is similar since this simply requires determining if there is at least one other possible winner of the tournament via manipulations.

Theorem 4.2 (Destructive Manipulation of Cups)**.** *Determining if a cup tournament can be destructively manipulated using tournament manipulations takes polynomial time.*

Proof. Determine the set of possible winners using Algorithm 4.1. If there exists a team other than the team the coalition wishes to lose then there is a destructive manipulation. □

Example 4.2. *Referring to Example 4.1, notice that the set of possible winners of the tournament can be at least two different teams. Therefore, the coalition can ensure that any team in the cup does not win if they so chose. For example if their goal was to make*



$\{1_0, 4_3, \mathbf{7}_3, 9_2, 12_2, 15_1, 16_3\}$
$\{1_0, 4_1, \mathbf{7}_2\}$
$\{9_0, 12_1, 13_2, 14_3, 15_1, 16_3\}$
$\{1_0, 4_1\}$
$\{5_0, 6_1, \mathbf{7}_1\}$
$\{9_0, 12_1\}$
$\{13_0, 14_2, 15_1, 16_2\}$
$\{1_0\}$
$\{3_0, 4_1\}$
$\{5_0, 6_1\}$
$\{\mathbf{7}_0\}$
$\{9_0\}$
$\{11_0, 12_1\}$
$\{13_0, 14_1\}$
$\{15_0, 16_1\}$
$\{1_0\}$ $\{2_0\}$ $\{3_0\}$ $\{4_0\}$ $\{5_0\}$ $\{6_0\}$ $\{\mathbf{7}_0\}$ $\{8_0\}$ $\{9_0\}$ $\{10_0\}$ $\{11_0\}$ $\{12_0\}$ $\{13_0\}$ $\{14_0\}$ $\{15_0\}$ $\{16_0\}$

Figure 4.1: The cup competition described in Example 4.1. There are sixteen teams and each coalition member is denoted with a filled circle. In the example, the coalition is attempting to make team 7 the winner, which is bolded throughout the diagram. At each node of the competition, there is a set of teams that corresponds to the possible winners of that node via manipulation by the coalition. The first team in each list is the expected winner of the match without manipulation and all other the teams represent those which could win the match given manipulation. Since team 7 is a possible winner of the final game at the apex of the cup, then it is possible for the coalition of 3, 5, 11, 12, 13 and 15 to manipulate the competition so that seven wins. The minimal number of manipulations needed to make a given team a winner at each level is denoted by the subscript on the team number.

Table 4.1: The adjacency matrix of the tournament graph for the cup described in Example 4.1. If a team has a 1 in their row then they win the game and, if not, they would lose.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	0	1	1	0	1	1	1	0	1	1	0	1	1	1	0	0
2	0	0	1	0	0	1	1	0	1	1	0	1	0	1	0	1
3	0	0	0	1	1	0	1	0	1	0	0	0	1	1	0	1
4	1	1	0	0	1	1	0	1	1	1	1	0	1	1	0	1
5	0	1	0	0	0	1	1	0	0	1	1	0	1	1	1	0
6	0	0	1	0	0	0	1	1	0	0	1	1	1	0	1	1
7	0	0	0	1	0	0	0	1	0	1	1	0	1	1	1	0
8	1	1	1	0	1	0	0	0	0	0	0	1	1	0	0	1
9	0	0	0	0	1	1	1	1	0	1	1	0	1	1	0	1
10	0	0	1	0	0	1	0	1	0	0	1	1	0	0	1	0
11	1	1	1	0	0	0	0	1	0	0	0	1	1	1	0	0
12	0	0	1	1	1	0	1	0	1	0	0	0	1	1	1	0
13	0	1	0	0	0	0	0	0	0	1	0	0	0	1	1	1
14	0	0	0	0	0	1	0	1	0	1	0	0	0	0	0	0
15	1	1	1	1	0	0	0	1	1	0	1	0	0	1	0	1
16	1	0	0	0	1	0	1	0	0	1	1	1	0	1	0	0

t_1 lose, then by the second round the coalition could have caused the loss by forcing t_1 to play team t_4.

4.2.1 Minimal Number of Manipulations for Cup Competitions

Having a coalition that can easily manipulate tournaments is dangerous but even more distressing is that teams can find the minimal set of manipulations needed to make the team the winner with very little extra effort. Given a pool of willing conspirators, it makes sense to select the smallest group of cheaters for two reasons. First, the fewer games that are manipulated the less suspicious a manipulation strategy. Second, the cost would almost assuredly go up and the split of the winnings earned by each member would be less. Thus, it is reasonable to assume that any cheater attempting to manipulate a competition would find the smallest available coalition to do so.

Computing the minimal number of manipulations simply requires keeping a count within Algorithm 4.1 for computing a manipulation. Let $s_\ell^{t_i}$ be the match at level ℓ where t_i is a leaf node of the sub-tree below $s_\ell^{t_i}$. The level is defined as the height from the bottom of the cup tree, which is assumed to be a perfectly balanced binary tree, and level 0 is the level belonging to the leaves.

Let c_{ij} be a constant that is 1 if $(t_j, t_i) \in M$ and 0 otherwise, where $M \subseteq E$ is the set of edges which can be manipulated by the coalition. This corresponds to $c_{ij} = 1$ when a manipulation must occur for t_i to win and 0 otherwise. The minimal number of manipulations needed to win a match $s_\ell^{t_i}$ is,

$$m(t_i, s_\ell^{t_i}) = \begin{cases} 0 & \text{if } \ell = 0 \\ m(t_i, s_{\ell-1}^{t_i}) + \min_{t_j \in D_i}(m(t_j, s_{\ell-1}^{t_j}) + c_{ij}) & \text{if } \ell > 0 \end{cases},$$

where D_i is the set of teams that t_i can defeat fairly or are a coalition members.

Lemma 4.1 (Minimal Number of Manipulations)**.** *The minimal number of manipulations needed to make a team t_i a winner at level ℓ in the tree is equal to $m(t_i, s_\ell^{t_i})$.*

Proof. By induction. **(Base Case)** First, observe that the minimal number of manipulations at a leaf is 0. Hence, $m(t_i, s_0^{t_i}) = 0$ for all leaves t_i. Next note that at level 1 there are only 2 nodes in the possible winner sets of the leaves. Therefore, if t_i can defeat t_j, $m(t_i, s_1^{t_i}) = m(t_i, s_0^{t_i}) + m(t_j, s_0^{t_j}) + c_{ij} = c_{ij}$ which is the exact number of manipulations that have occurred to make t_i a possible winner so far. **(Induction Step)** Assume the premise for $1 < \ell \leq k$. Now, $m(t_i, s_{k+1}^{t_i}) = m(t_i, s_k^{t_i}) + \min_{t_j \in D_i}(m(t_j, s_k^{t_j}) + c_{ij})$. It is given that $m(t_i, s_k^{t_i})$ is the minimal number of manipulations needed to make t_i a winner at level k by the assumption and, for every $t_j \in D_i$, it is known that $m(t_j, s_k^{t_j})$ is also the minimal number of manipulations needed to make each t_j a winner at level k. By definition, c_{ij} is the number of manipulations for t_i to defeat t_j. Since t_i can defeat any t_j in D_i, the one with the fewest previous manipulations to be a winner at level k plus c_{ij} leads to the fewest manipulations in total to make t_i win the sub tournament $s^{t_i}_{k+1}$. This equals the minimum over the set D_i. Therefore the lemma holds for $k+1$ and, by induction, all ℓ levels of the tree. □

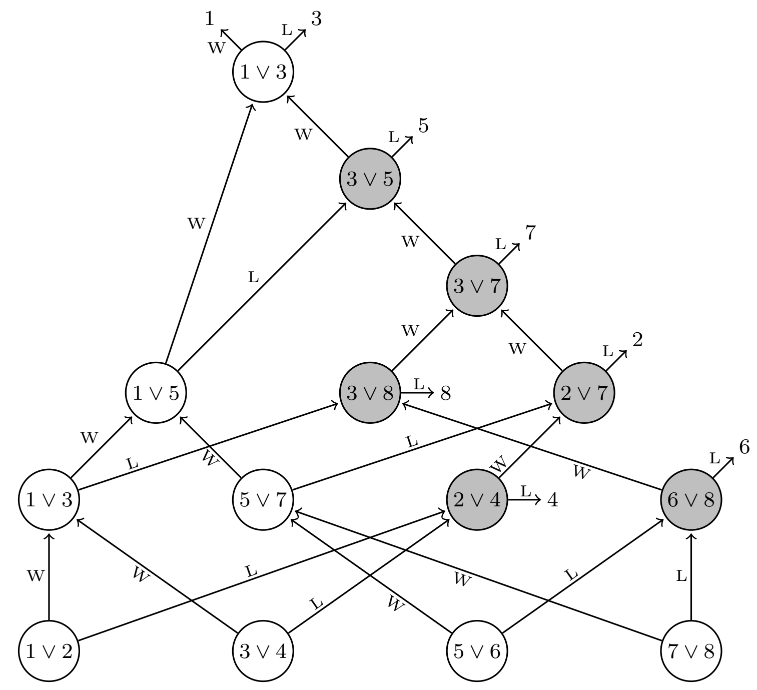
Theorem 4.3 (Minimal Manipulation is Polynomial)**.** *A modified CSL algorithm, where the team which minimizes the value of $m(t_i, s^{t_i}_n)$ is selected to lose to team t_i at every node $s^{t_i}_n$, calculates the minimal number of manipulations needed to constructively or destructively manipulate a cup competition in polynomial time.*

Proof. By Lemma 4.1, the value of $m(t_w, s^{t_w}_n)$ at the root node is the minimal number of manipulations which ensures t_w is the winner. All that remains is to show that the algorithm remains polynomial. The modified CSL algorithm still makes $O(m^2)$ comparisons. The only difference is that it is necessary to calculate the minimum which can be done by storing the minimum as each team is checked. Therefore, the time complexity remains $O(m^2)$. Constructive manipulation requires calculating $m(t_w, s^{t_w}_n)$ whilst destructive manipulation requires the minimum over all other teams. □

Example 4.3. *Referring to Example 4.1, there are a variety of ways that t_7 could be made a winner using more or fewer manipulations. This example illustrates how to calculate the minimal set of manipulations needed to make t_7 the winner. Figure 4.1 shows the cup tree annotated with both the teams and the minimum number of manipulations needed to make each team a winner. For example, in the final round, t_7 could play t_{12}, t_{13} and t_{15}. t_7 needs two manipulations to get to that round. t_{13} also needs two manipulations but loses to t_7 and a total of four manipulations is required. However, t_7 can defeat t_{15}, who only need one manipulation, so only three manipulations are required. Since t_7 loses to t_{12} and would require a manipulation, four manipulations are required. So t_7 plays t_{15} and would need a minimum of three manipulations from t_3, t_5 and t_{13}. Coalition members t_{11}, t_{12} and t_{15} are unnecessary. Note that the coalition can make t_{15} the winner with only a single manipulation which means destructive manipulation is easier and requires only t_{13}.*

4.2.2 Double Elimination Cup Competitions

In a double elimination competition, a manipulation of the tournament does not automatically bounce the manipulator out of the tournament as in the single elimination case. Recall that a double elimination tournament is a cup where each team must lose two games to be eliminated. Figure 4.2a shows an example of a double elimination tournament with eight teams. Since teams are not immediately eliminated, a manipulator may be able to



(a)

	1	2	3	4	5	6	7	8
1	0	1	1	0	1	0	0	1
2	0	0	0	1	0	0	1	0
3	0	1	0	1	1	0	1	1
4	1	0	0	0	1	1	1	1
5	0	1	0	0	0	1	1	0
6	1	1	1	0	0	0	0	0
7	1	0	0	0	0	1	0	1
8	0	1	0	0	1	1	0	0

(b)

Figure 4.2: **(a)** An eight team double elimination cup. The shaded nodes represent the matches where both teams have lost at least one match and could be eliminated. **(b)** The adjacency matrix for the tournament graph related to the double elimination cup, where a 1 in position (i, j) denotes that team t_i defeats team t_j in a fair match.

manipulate the tournament twice. As well, a single team could manipulate the tournament and still earn a victory in that tournament. Unfortunately, the CSL algorithm does not work in this case because a manipulator can take more than one path through the tournament. Given that coalitions are likely to be small in practice, a polynomial algorithm for double elimination tournaments if the size of the coalition is constant is shown. While this algorithm is not necessarily applicable in all situations, it could be argued that, for the coalitions likely to be found in practice, this method could be used.

Lemma 4.2 (Double Elimination Manipulation)**.** *For double elimination tournaments, if the size of the coalition is of bounded size c, determining whether there is a constructive manipulation takes polynomial time.*

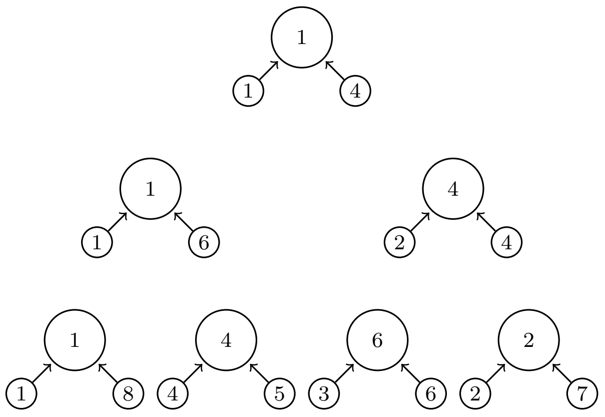
Proof. If there is a coalition of bounded size c then a team can manipulate the cup only once if they wish to win the tournament and twice if the team desires another team to win. At each step in the tree, a team must decide whether to manipulate or not. Since there are two chances to be eliminated, there remains c teams after any coalition member manipulates once. Only after they have manipulated a second time are they removed from the competition. This means there are at most 2^c different manipulation strategies at each of the $log(m)$ levels. This gives $O(2^{\log(m)c})(= O(m^c))$ possibilities that can be checked in linear time, which gives a polynomial algorithm for determining if there is a constructive manipulation. □

Example 4.4. *To illustrate how a manipulation strategy could be determined for double elimination cups, referring to Figure 4.2, assume there is a coalition consisting of t_2 and t_8 that are attempting to make t_4 the winner. Both t_2 and t_8 are expected to lose their first game and thus cannot manipulate that game. However, both win the next game so there are four possible options in terms of manipulation; i.e., every pairing of manipulating or not for both teams. However, t_2 must manipulate their game against t_4 or their desired winner would be eliminated. One possibility is that t_2 manipulates in the second round and no other manipulations occur. t_4 would win the next game but end up losing to t_3, which is not a valid manipulation strategy. Therefore, t_8 must also manipulate their second game and thus both coalition members would be eliminated. Fortunately, t_6 beats t_3 when t_8 throws the game and t_4 wins its four remaining games. Therefore, there is a valid strategy where both coalition members manipulate matches in the second round.*

4.2.3 Reseeding Cup Competitions

Another variant on the cup that is used in practice is those cups that have reseeding within them. There are two common kinds of reseeding cups. The simplest kind is random reseeding. This means that after each round a random draw is held to determine opponents. One example of this is in soccer where the final eight of the EUFA Champions League are seeded via a draw. Another common type of reseeding is ranked reseeding where reseeding is based on rank where teams are reseeded to ensure top teams gain an advantage at each round. This type of reseeding is used in the National Hockey League playoffs. Figure 4.3 shows a cup using ranked reseeding.

In general, the complexity of determining if a valid manipulation exists in these mechanisms is unknown. However, a sub class of these problems where the coalition size is fixed is polynomial. The CSL algorithm cannot be applied because teams could have multiple paths through the tournament. However, if the size of the coalition is of bounded size c, then a manipulation strategy can be found in polynomial time. As in Section 4.2.2, coalitions are likely to be of a small bounded size and this method might be applicable in practice.



(a)

	1	2	3	4	5	6	7	8
1	0	1	1	1	1	1	0	1
2	0	0	0	1	0	0	1	0
3	0	1	0	1	1	0	1	1
4	0	0	0	0	1	1	1	1
5	0	1	0	0	0	1	1	0
6	0	1	1	0	0	0	0	0
7	1	0	0	0	0	1	0	1
8	0	1	0	0	1	1	0	0

(b)

Figure 4.3: **(a)** A ranked reseeding cup. Teams are ranked from 1 to 8, with 1 being the top ranked team. After the results of each round are found, the pairings for the next round are generated while ensuring the top ranked team remaining plays the worst ranked team remaining and so on. **(b)** The adjacency matrix corresponding with the ranked reseeding cup.

Lemma 4.3 (Reseeding Cup Manipulation)**.** *For a ranked reseeding cup competition, if the manipulating coalition is of bounded size c, then determining a set of manipulations that makes a team win takes polynomial time.*

Proof. The key observation is that with a constant-sized coalition there are only a polynomial number of ways to manipulate the games by rearranging the tournament graph. It suffices to check the winner of each of the polynomial number of fixed tournament graphs. For each fixed tournament graph, the winner can be determined in linear time as there are only $O(m)$ matches to check.

At most c of the $\frac{m}{2}$ matches in the first round have more than one team as a possible winner. This means that there is at most 2^c possibilities to examine after each round. As there are $\log(m)$ rounds, at most $(2^c)^{\log m}$ ($=m^c$) possibilities must be considered. Given $O(m^c)$ possible manipulation strategies for an unfixed cup with ranked reseeding and a bounded sized coalition, it is sufficient to check each arrangement, which can be done in linear time. This gives a polynomial algorithm for bounded c. $\square$

Example 4.5. *Figure 4.3a shows a reseeding cup where each game happens according to the tournament graph (Figure 4.3b). Now assume that t_2 and t_4 are conspiring to manipulate the tournament so that t_5 wins the cup. t_1 only loses to t_7 so t_2 must manipulate to t_7 in the first round otherwise t_1 wins the cup. As well, if t_4 beats t_5 in the first round then*

t_5 is eliminated and the goal cannot be achieved. Therefore, in this case, both teams must manipulate in the first round. t_5 would then play t_6 and t_7 in consecutive rounds defeating both of them to win the cup.

With random reseeding the problem can be separated into two issues: determining whether manipulation is possible to make a team a winner under every possible seeding and determining if there exists any seeding such that the coalition can manipulate the games to make a given team the winner. These problems are discussed in more detail in Section 4.4 as they are equivalent to seeding manipulation. Vu et al. [67] and Hazon et al. [28] tackle some probabilistic variants of possible winners without manipulation of games. However, the complexity of determining possible winners with a win-loss tournament graph in balanced cup trees remains open [35, 28, 52].

4.3 Round Robin Competition

Round robins are widely used in many real world tournaments. One advantage from an organizers point of view is that teams are not immediately eliminated if they lose a single game. However, this increases the susceptibility to cheating as teams are not necessarily eliminated by manipulating a game. One complication is that games do not always need to end in a victory and a variety of different scoring models can be used.

A *scoring model* is defined as a set of tuples giving the possible outcomes of a game. Copeland scoring has a simple win-loss ($\{(0,1),(1,0)\}$) scoring model where the winning team earns one point and the losing team earns none. Bartholdi, Tovey and Trick [10] show that constructive manipulation can be determined in polynomial time for a chess scoring model ($\{(0,1),(\frac{1}{2},\frac{1}{2}),(1,0)\}$). Faliszewski et al. [22] show that for a range of scoring models manipulating Copeland voting, i.e. a round robin election, is NP-Complete. The full characterization of the polynomially manipulable scoring models is made in this chapter.

The problem of manipulating a round robin is the problem of determining which games need to be manipulated to ensure that a given team t_w wins the competition. Given that coalitions are assumed to only be able to throw games they should have won, there are some games that cannot be affected by the coalition and are fixed. All other games are manipulable. Games between coalition members can earn any of the possible scores allowed by the scoring model. Games where coalition members and non-coalition members compete are restricted so the manipulator earns equal or less points and the non-member earns equal or more points. To fully characterize the set of polynomial scoring models, a correspondence between manipulation in round robins and winner determination problems is made. Using this observation, the following theorem is obtained.

Theorem 4.4 (Constructive Manipulation for Round Robins)**.** *Determining if there exists a constructive manipulation of a round robin competition is solvable in polynomial time if the scoring model, normalized as described by Kern and Paulusma [34], is of the form* $S = \{(i, n-i) \mid 0 \leq i \leq n\}$*, and NP-complete otherwise.*

Proof. First observe that in manipulation problems, there exists games which can be manipulated and games which cannot be manipulated. In winner-determination problems, there exists games which have been played and those remaining to be played. In both cases, the goal of the problem is to find an assignment of the changeable games so that a specific team wins the tournament. In manipulation problems, however, the possible set of scores between a non-coalition member t_i and a coalition member t_j are restricted but it can be shown that the restricted model is still a conforming model—a model that is of the form $S = \{(i, n-i) \mid 0 \leq i \leq n\}$—when normalized. Assume there is a normalized, conforming scoring model and the initial result of the game is (c_i, c_j), then the remaining valid scores that can be assigned are those from (c_i, c_j) to $(n, 0)$ as the non-coalition member cannot have a lower score given the restriction. By normalizing this new model, a new model is obtained, $\{(0, c_j), \ldots, (n-c_i, 0)\}$ which is conforming. Note that it is possible for an initially non-conforming model to become conforming for games between non-coalition members and coalition members. However in games between two coalition members, any outcome is possible and, therefore, the scoring model would still be non-conforming as a whole. Kern and Paulusma [34] showed that determining if a team can win a tournament (i.e. is not eliminated from competition) takes polynomial time, using flow networks, if the normalized scoring model is of the form $S = \{(i, n-i) \mid 0 \leq i \leq n\}$ and is NP-complete otherwise. □

Example 4.6. *To illustrate the concept of manipulating the round robin tournament, assume there exists a six team single round robin, the tournament uses a scoring model with only wins and losses and the tournament graph, shown in Table 4.2, represents the outcome of all fair matches in the competition. Now suppose a team or group of teams wishes to make* t_6 *the outright winner. As it stands,* t_6 *is tied with* t_1 *with four wins if all games are played fairly. In this example, the conspirators could only be* t_1 *or* t_5 *as none of the other teams can cause* t_6 *to win the tournament. First, if* t_5 *was the conspirator,* t_5 *could lose the game against* t_6*, making* t_6 *the winner. Second, if* t_1 *was the conspirator having already lost to* t_6*,* t_1 *must lose a game to another team without making the new victor of the manipulated game the winner. Clearly, in this example, it can be seen that* t_1 *could safely lose one game to* t_2*,* t_3 *or* t_4*. For illustrative purposes, the exact flow graph needed to solve these problems in general as described by Kern and Paulusma [34] is shown in Figure 4.4. The graph shows a matching problem where games are assigned winners so that total number of victories for any team is less than those of* t_6*, who earned four victories. Note that* t_5 *can earn one less victory since* t_5 *wins when playing* t_6*, which is not represented in the graph.*

Table 4.2: The tournament graph, as an adjacency matrix, for Example 4.6.

	1	2	3	4	5	6
1	0	1	1	1	1	0
2	0	0	1	0	0	0
3	0	0	0	1	0	0
4	0	1	0	0	1	0
5	0	1	1	0	0	1
6	1	1	1	1	0	0

Using Theorem 4.4, it is possible to answer, at least for the direct tournament manipulation case, the three open cases left by Faliszewski et al. [22]. The three remaining scoring instances left open in Faliszewski et al. [22] are $\{(0,1),(0,0),(1,0)\}$, $\{(0,1),(1,0),(1,1)\}$ and $\{(0,1),(\frac{1}{2},\frac{1}{2}),(1,0)\}$. The normalized form of these scoring models are $\{(0,0)\}$, $\{(0,1),(1,0)\}$ and $\{(0,2),(1,1),(2,0)\}$. Problems using these models can be solved in polynomial time as they are all of the form $S = \{(i, n-i) \mid 0 \leq i \leq n\}$.

By comparison, it is always polynomial-time solvable to determine if a destructive manipulation exists.

Theorem 4.5 (Destructive Manipulation of Round Robins)**.** *Determining if there is a destructive manipulation of a round robin competition takes polynomial time.*

Proof. Assume that t_l is the team that the coalition desires to lose. It is sufficient to check whether the maximum points of another team via manipulation is greater than the points of t_l. If t_l is a member of the coalition and therefore a manipulator, for each team t_i that is checked, only manipulations that increase the relative points between t_i and t_l are applied. For all other teams, the manipulations which decrease the points of t_l or increase the points of t_i are applied. If t_l is not a member of the coalition, no games involving t_l may be manipulated since manipulations are restricted to allow only those manipulations that increase the points of t_l and, therefore, increase the relative gap between t_l and the manipulator. Therefore, no other team is better off when games involving t_l are manipulated. In both cases, the manipulations that increase the points of the team under consideration against all other teams is applied. If the total number of points of any other team is greater than the points of t_l under these manipulations, then there is a destructive manipulation of t_l. This algorithm can be run in $O(m^2)$ time. □

Example 4.7. *To illustrate the concept of destructively manipulating round robin tournaments, recall the example described in Example 4.6. Suppose the conspirators wish for t_1 to not win the tournament. The only team that could change the result is t_5 by losing to t_6 and making t_6 the outright winner with five victories over t_1's four victories.*

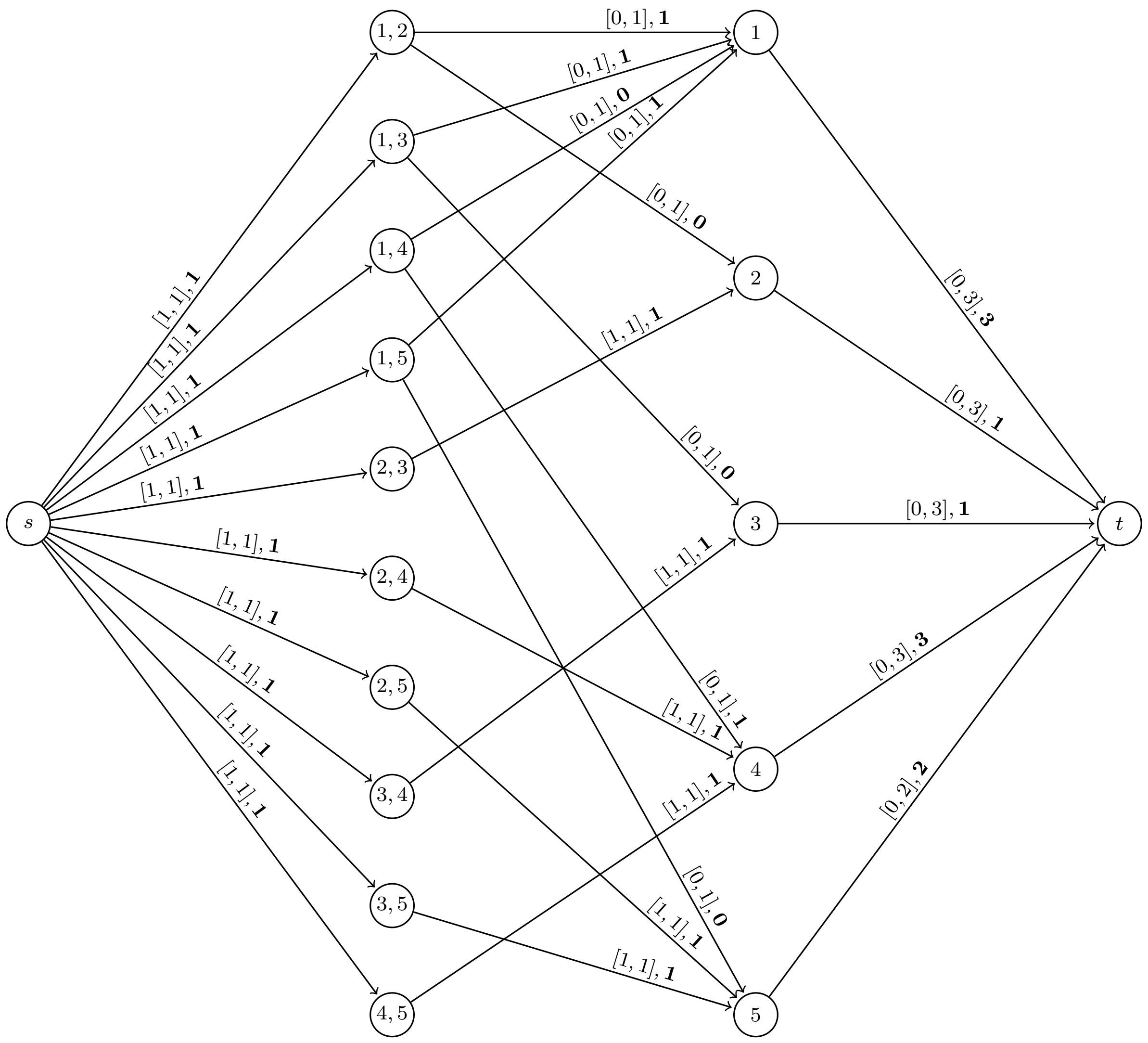


Figure 4.4: The feasible flow network from Example 4.6. The edges are denoted with upper and lower bound capacities in the brackets and a corresponding feasible flow, shown in bold. The conspirator (Team 1) can change each of the games they were supposed to win but all other games are fixed.

To extend these results to the multiple round tournaments, it suffices to just add the extra games to the flow graph used to calculate winner determination problems as described by Kern and Paulusma [34]. For a round robin with n rounds, there is an additional $(n-1)\frac{m(m-1)}{2}$ nodes and $3(n-1)\frac{m(m-1)}{2}$ edges in the feasible flow graph, where m is the number of teams in the round robin.

A further complication is when the goal of manipulation is just to earn a berth in the next round of the playoffs. One type of this problem was discussed in Chapter 3. It is NP-Complete to decide these questions under most playoff systems for all scoring models [45, 27]. Specifically, if there are more than two teams making the playoffs or if there are wild card teams then the problem is NP-Hard to determine who wins a tournament. As discussed before, manipulation problems are qualification and elimination problems because unmanipulable games can be mapped to played games and those games which can be manipulated are mapped to games remaining. This simple reduction shows that the problems are identical and that the manipulation problems are also NP-Complete.

4.3.1 Minimal Number of Manipulations for Round Robin Competitions

Like the manipulations in cup competitions, it is unlikely that coalitions would expose themselves to extra risk by manipulating more games than necessary. In this section, it is shown that determining the minimal number of manipulations for round robin competitions with the win-loss scoring model can be done in polynomial-time. Only win-loss scoring is considered in this section but it is conjectured that similar methods could be developed for other scoring models, specifically those scoring models that were polynomial-time solvable in Section 4.3.

The goal is to find the minimal set of manipulable edges that need to be changed so that the desired team t_w has at least as many wins as any other team in the competition. Before it is shown how to calculate the minimal number of manipulations, a greedy method is presented for determining the number of wins that t_w would earn from coalitional manipulation while preserving minimality. The intuition for this greedy algorithm is that selecting manipulations to increase the wins of t_w, if possible, always benefits t_w while reducing the number of wins of only the coalition members. Each coalition member which could lose to t_w by manipulation is added to a set. While there are coalition members that can lose games to t_w and have more wins than t_w, remove the coalition member with the most wins and reverse the edge in the tournament. The reversed edge is a minimal manipulation and is stored. Recalculate the number of wins based on the newly adjusted tournament graph and repeat the procedure until either there are no teams in the set or every team has equal or less wins than t_w.

Lemma 4.4 (Greedy Manipulations are Optimal). *Algorithm 4.2 determines the minimal number of manipulations needed to find the number of wins earned by t_w in any minimal set of manipulations.*

Proof. First, recall that t_w can earn no extra wins except by manipulation by coalition members so the final number of wins earned by t_w can only change if the coalition manip-

```
Algorithm: GreedyPreprocess($G$, $S$, $t_w$)
input  : A Tournament Graph $G = (T, E)$, a coalition of teams $S$ and a team $t_w$.
output: The minimum number of manipulations needed to find the number of wins
        earned by $t_w$ in any minimal manipulation strategy.
for every team $t_i$, calculate the number of wins, $w_i$, according to $G$;
remove all teams from $S$ which lose to $t_w$ according to $G$;
while $\exists_{t_i \in T}\ w_i > w_w$ and $S$ is not empty do
    find the $t_i \in S$ with the largest $w_i$;
    remove $t_i$ from $S$;
    reverse the edge $(t_i, t_w)$ in $G$;
    add $(t_w, t_i)$ to Manipulations;
    recalculate, for every team $t_i$, the number of wins, $w_i$, according to $G$;
return Manipulations;
```
Algorithm 4.2: The algorithm for preprocessing the problem of finding the minimum number of manipulations needed to make a team the winner.

ulates games so that t_w wins. Next, it is necessary to show that increasing the wins of t_w is at least as minimal as any strategy. Observe that to reduce the number of wins of two or more nodes that have more wins than t_w it requires two manipulations while increasing the number of wins of t_w requires only a single manipulation. While it does only require a single manipulation to decrease the number of wins of one node, another node, which is not necessarily t_w, must earn extra wins. Therefore, increasing the score of t_w is most efficient, if possible.

For this to be a minimal number of manipulations, the greedy algorithm must not increase the number of wins of t_w unnecessarily. Assume that more than the minimal number of manipulations was used by the greedy algorithm. This means that an edge was selected that did not decrease the team with the most wins of the conspirators or further edges were selected even though the number of wins of t_w was already higher than all other teams. However, since the coalition member with the most wins was selected and subsequently reduced, it is not possible to have increased the number of wins unnecessarily in this manner. As well, the stopping condition assures that the algorithm stops whenever the number of wins of t_w is equal to or higher than all other nodes. Therefore, a contradiction is found and the greedy algorithm uses only a minimal number of manipulations upon termination. □

Example 4.8. *In Example 4.6, it was determined that t_5 could lose their game against t_6, making t_6 the winner. However, t_5 also could have lost their games to t_2 and t_3. Only the single loss to t_6 is minimal according to Lemma 4.4 because the stopping condition, that t_6 had more victories than every other team, was reached. If the coalition was t_1, then no greedy manipulation is possible and the wins earned by t_6 remains at four.*

If the greedy algorithm does not successfully make the team t_w the winner because they could not lose enough games to t_w in order to increase their points beyond every other team then the coalition must do more work to make the distinguished team the winner. This is calculated using a minimum cost feasible flow algorithm. As in Section 4.3, a winner can be determined using a feasible flow algorithm but this strategy may have used more manipulations than necessary. A minimum cost feasible flow algorithm can be calculated in polynomial-time if the weights on the edges are multiplied by the flow values [24]. The minimal cost solution is the one where the least number of edges in the tournament graph are reversed and manipulations are represented as the edges in the flow graph from a node (t_i, t_j) to t_i where $(t_i, t_j) \in T$ and $t_i \in S$ where S is the coalition. By giving a weight of 1 to each edge representing a manipulation, the solution incurs a cost of 1 for each edge it uses. Therefore, the minimal cost feasible flow is the solution which has flow along the fewest edges representing manipulations.

Theorem 4.6 (Minimal Manipulations of Round Robins)**.** *Determining the minimal number of tournament manipulations given a tournament graph $G = (T, E)$ required with a win-loss scoring model to make t_w a winner, if possible, takes polynomial-time.*

Proof. Let c be the number of wins of the distinguished node, t_w, calculated using Algorithm 4.2. If the stopping condition has not been reached, c is used to determine how many more manipulations are necessary. A winner determination flow network is constructed as described by Kern and Paulusma [34] and Gusfield and Martel [27] (see Figure 4.4, for an example). A weight of 1 is added to each edge (t_i, t_j) where $(t_i, t_j) \notin E$ and therefore represents a manipulation. All other edges have the weight 0. The feasible flow which uses the fewest of the non-zero edges is the minimal number of tournament manipulations to achieve a constructive manipulation. Since the value of c can be determined in a linear number of steps and a single minimum cost flow computation, which is polynomial, is needed to determine the remainder of the minimum number of manipulations necessary to make t_w the team with the most wins, if possible, the entire procedure can be calculated in polynomial-time. □

Example 4.9. *In Example 4.8, t_5 was conspiring to change the outcome of the tournament. t_5 successfully achieved this using only the greedy manipulations. t_1 was the second coalition described in Example 4.6. t_1 lost its only game to t_6 fairly and thus nothing is gained by applying the greedy manipulations and the number of wins of t_6 remains at four. Therefore, t_1 must lose either the game against t_2, t_3 or t_4. The obvious minimal choice is to choose just one of the games to lose as this would satisfy the conditions and t_6 would be the victor. Suppose, for the purpose of illustration, that this was not immediately clear and t_1 had to choose the best subset of possible manipulations to achieve its goal. The flow network described in Figure 4.4 has two edges for each game t_1 plays representing the possible choices of t_1. By adding a cost of 1 for each game that could be manipulated, specifically the edges*

((1,2),2), ((1,3),3) and ((1,4),4), and having no cost for every other edge, the minimal cost feasible flow on the network returns a valid minimal solution. Note the feasible flow shown in bold is also a minimal cost flow.

4.4 Seeding Cup Competitions

To this point, the focus has been on coalitions of teams manipulating the result of a competition. However, it is possible for the scheduler of a competition to create a seeding that favours one team over other teams. This section discusses some of the various ways in which a seeding can be manipulated. While the previous two sections on cups and round robins have focused on theoretical results, the theoretical complexity of this problem, when the competition is balanced, remains open. Therefore, the focus shifts to how difficult these problems can be in practice, regardless of the complexity. The results of this chapter suggests that if these problems are theoretically difficult then it provides very little in terms of a safety guarantee as large problems using hundreds of teams can be solved in less than a second.

A competition must be organized by some mechanism and this mechanism may be susceptible to manipulation by the scheduler of the tournament. The scheduler is referred to as a single entity but in reality may be a committee of individuals. In this section, the various vulnerabilities of the schedule of a cup competition, commonly called a seeding, are highlighted. The standard seeding manipulation problem, highlighted in previous research [35, 28, 67], represents the most unrestricted problem and therefore is the most open to manipulation. A set of realistic restrictions are added to the problem and it is shown that even under restricted conditions manipulation strategies are often easy to find.

4.4.1 Definitions

The *seeding* of a competition specifies which teams must play each other at every round of the cup, assuming a team advances. Another possible view of a seeding is an assignment of teams to the leaves of a cup tree. These seedings could be generated by a random draw but it is also possible that a scheduler generates the seeding. A *seeding manipulation strategy* is any set of deliberate scheduling actions in an effort to cause the desired team, t_w, to win the competition. A *scheduling action* is a placement of a team in the seeding. Not discussed in the canonical description of this problem [35, 28, 67, 70], there are restrictions which could be placed on the scheduler of the tournament. While some of these restrictions ensure that less manipulation can occur, the only way to remove the possibility manipulation is to fully remove the scheduler from the process as is done in a random draw. Note that a random draw is often undesirable as this can lead to two favourite teams facing each other early in the tournament instead of in the finals. To move away from a random draw, restrictions are added to ensure that the seeding conforms to the desires of the organizers like, for example, that top teams do not face each other immediately.

Four different types of restrictions can be added to the scheduling process. A *pooling restriction* requires that a prespecified set of teams—called a pool—play each other before playing other teams. A *team arrangement restriction* requires that the positioning of teams must meet some criteria—a set of rules and conditions—when setting the schedule (see [48] for an example of real world criteria). A *pool arrangement restriction* requires that the winners of pools must be positioned according to some criteria. A *criteria modification restriction* requires that the criteria by which teams are arranged is fixed and known. Example 4.10 discusses how these restrictions are applied to a real world problem.

Example 4.10. *The NCAA Division One Basketball Championship, commonly called March Madness [48], is held annually in March and April. The championship uses a single elimination cup structure where 64 teams are seeded in a balanced cup tree. The teams are separated into pools of sixteen teams. Each team is assigned a rank from 1 to 16 and only one team of each rank may belong to a pool. The cup has four pools of teams but the makeup of the pools is made by the scheduler [48]. If the scheduler so chose, pools could be generated so that the scheduler's desired team would belong to a pool where most of the teams would lose to the desired winner. Therefore, there is no pooling restriction on the pools in the NCAA tournament. An example where pools are restricted would be if they were geographically based, like the NHL playoffs. Figure 4.5a shows how teams are arranged in a pool so the top teams play weak teams as this competition has a team arrangement restriction. If this arrangement was not fixed, then it would be possible to have the top ranked team play the second ranked in the first round of the tournament and then face the winner of the third and fourth ranked team in the next round. This hypothetical arrangement of teams, shown in Figure 4.5b, guarantees that three of the top four teams in the pool are eliminated after the second round. March Madness championships do not have a pool arrangement restriction [48]. As such, the scheduler is able to arrange the final two rounds as they see fit. The possible arrangements of the four pools of the NCAA Championship are shown in Figure 4.5c. In a competition with a pool arrangement restriction, there would be some predetermined method for arranging the pools for the final two rounds. An example of such a method would be if the pools were geographically based and the Northwest always played the Southwest in the semi-finals and then played the final team from the East in the finals. Since the ranking of the teams is decided by the scheduler [48], there is no restriction on criteria modification and the scheduler can use this to change the seeding [17]. The imprecise nature of the rank generation allows for weaker teams to be boosted in rank and top teams to be diminished. While it is unlikely that wholesale changes would be made to the rank, it would be possible to modify the rank of teams a little while still gaining a large advantage. Suppose that a team has an actual rank of eight, under the*

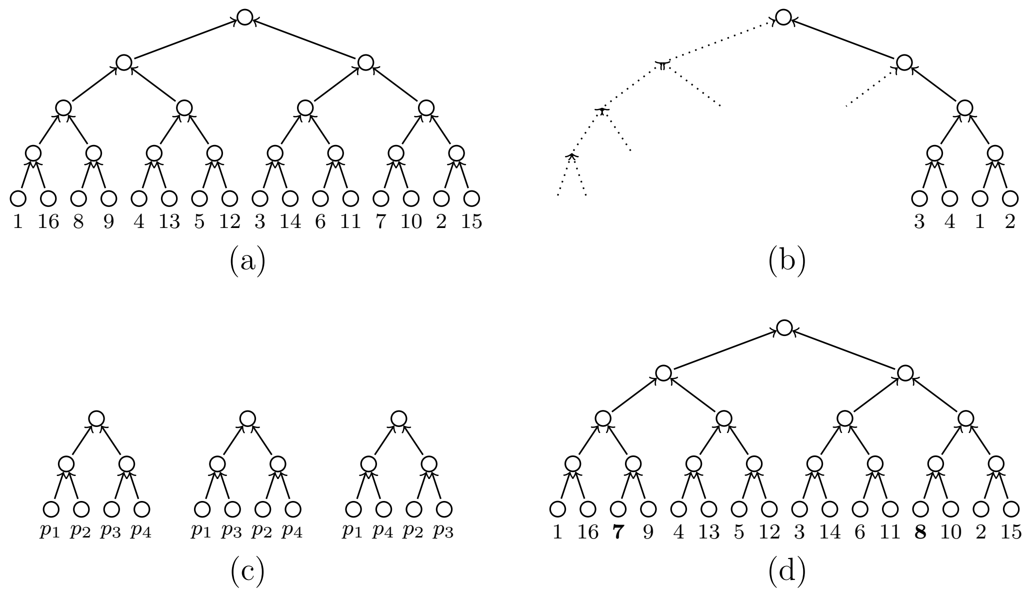


Figure 4.5: **(a)** The arrangement of teams by rank used by the NCAA Division One Basketball Championship. The numbers below the leaf nodes represent the rank of the team at that leaf. The arrangement is designed so that if the top ranked team always wins the arrangement preserves the property that the top ranked team plays the weakest ranked team. **(b)** A seeding can be manipulated so that all of the top teams play each other in the first two rounds. **(c)** The possible ways that the winners of the four pools of the NCAA could be arranged. **(d)** Changing the rank of a team from eight to seven can completely change their opponents.

arrangement described in Figure 4.5a, it is possible to completely change which opponents the team plays by increasing its rank by one as a team of rank seven plays on the other side of the cup tree (see Figure 4.5d).

Given the different restrictions, it is possible to construct families of restrictions and classify competitions based on the restrictions present. As a naming convention, a family of restrictions is considered as a four tuple where each tuple value can be either 0 or 1 depending on the presence of the restriction. An example tuple for the family without any restrictions would be 0000. The tuple placements represent, from left to right, the pooling restriction, the team arrangement restriction, the criteria modification restriction and the pool arrangement restriction. Families that are equivalent regardless of a restriction being present are merged and the corresponding tuple value is replaced with a don't care value or X. The complete set of families along with the restriction relationship are shown in Figure 4.6. An arc from one family to another represents that a restriction was added to the source

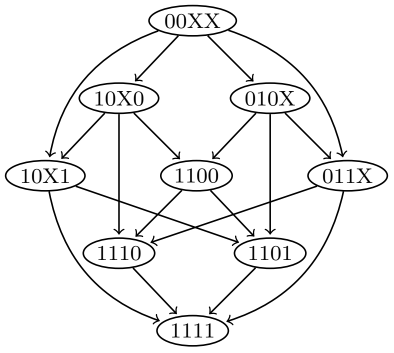


Figure 4.6: The various different possible configurations of elements of a seeding. Each label is a four tuple where 1 means fixed, 0 means unfixed and X is a don't care value. Don't care values are used to denote when the models are solution equivalent. The four tuple values represent are, from left to right, pooling, team arrangement, criteria modification and pool arrangement.

family to produce the new family. The *solution set* is the set of all possible solutions to the problem of finding a seeding manipulation strategy given a family of restrictions. Two families are equivalent if the solution set is identical. To formalize the notion of model equivalence, the two following lemmas are given.

Lemma 4.5 (Rank Modification Needs Fixed Team Arrangment)**.** *Any two families of restrictions with unfixed pooling and equivalent values for criteria modification and team arrangement have the same solution set regardless of the value of the pool winner arrangement.*

Proof. Let there exist two families $0ar0$ and $0ar1$ where a and r can be either value. Note that no pooling restriction means that any pool can be generated that obey the constraints of the values of a and r. The first family of restrictions allows pools to be arranged in any order and the second requires the arrangement be fixed. Clearly, any solution under the second family of restrictions is a solution to the first family of restrictions as the first allows any arrangement including the restricted arrangement. That each solution under the first family of restrictions is a solution under the second family restrictions can be seen by realizing the pools are unfixed. Any solution under the first family of restrictions can be transformed into a solution for the second family of restrictions by renaming the pools so the pools match the fixed arrangement which is allowed since the pooling is unrestricted in both cases. □

Lemma 4.6 (Pool Winner Arrangements Needs Fixed Pools)**.** *Any two families of restrictions with unfixed team arrangement and equivalent values for pooling and pool winner arrangement have the same solution set regardless of the value of the criteria modification.*

Proof. Assume that there are two families $p00w$ and $p01w$ where p and w can be either value. Now if the team arrangement is unrestricted then teams can be placed in any position that does not violate constraints specified by p and w. The first family of restrictions allows that teams can have ranks other than their actual rank and the second family of restrictions requires that teams must maintain their actual rank. Any solution under the second family of restrictions is a solution under the first family of restrictions as the rank does not necessarily need to be changed. Any solution under the first family of restrictions is a solution for the second family of restrictions since neither family of restrictions requires that a team of a given rank to play a team of a specific rank. Therefore, regardless of the rank assigned, any solution which obeys the constraints of p and w is a solution. □

Figure 4.6 also conveys the solution dominance relationship between the various models. An arc between two families of restrictions states that the family at the source of the edge contains all solutions of the model at the sink of the edge for any given problem. Lemma 4.7 proves this statement formally.

Lemma 4.7 (Dominance of Seeding Models)**.** *Given two families of restrictions A and B where family B contains only restrictions of A, $sol(B) \subseteq sol(A)$ where $sol(X)$ is the solution set for a family X.*

Proof. Given family B contains only restrictions of model A, A contains no restrictions that are not also in B. Thus, a solution under the restrictions of B can not violate a restriction in A and any solution under B is also a solution under A. □

4.4.2 A Description of the Families of Restrictions

Each restriction of the tournament implies that the choice is fixed without input from the scheduler. While it is possible to fully restrict a schedule, the result may be undesirable. For example, if geographic constraints are used to ensure unmanipulable pooling then top teams may be paired in the early rounds since they happen to be geographically close. In this section, each family of restrictions is described and, if possible, a real world example which uses these restrictions is described. It is shown that the families of restrictions cover a large number of different sports.

The most restricted family is 1111. Under this family of restrictions, all of the restrictions are enforced and therefore there is a single possible winner given the tournament graph. Competitions with these restrictions arise in the playoffs of many North American sports where the ranking is determined by points and some well defined set of tie breakers, the team arrangement is predetermined ahead of time, the pooling is based on some geographic or league feature and pool arrangement is irrelevant due to the fact there are only two pools.

The next most restricted family is the 1101 family where only the criteria modification restriction is not enforced. An example of this would be professional tennis where the ranking of players in the tournament do not necessarily correspond to the World Ranking of the same players. For example, players which have a specialty on a particular surface, like clay or grass, can have a higher rank in the seeding than their actual ranking.

While a very restrictive family, 1110 presents an interesting case. This states that the pooling, team arrangement and criteria are all immutable but the pool arrangement is not restricted. This is interesting because in competitions where there are just one or two pools then the problem corresponds to a 1111 family, the fully restricted case, but if there are more than two pools the problem is unconstrained in terms of pool winners. When the pooling, team arrangement and criteria are restricted, the pool winner can be determined from the tournament graph. Since the arrangement of the pool winner is unfixed, the winners can be manipulated into any arrangement that suits the scheduler corresponding to the $00XX$ family of restrictions. It is not known whether there exists any real world problems that have these restrictions.

The 1100 family of restrictions requires that the team arrangement and pooling of teams is fixed but the ranking of teams can be modified and the arrangement of pools is unfixed. Since professional tennis tournaments only have a single pool of teams, this family of restrictions could be applied to tennis as well.

An interesting case for reasons discussed in more detailed later, the $011X$ family of restrictions allows the pools to be unfixed but the team arrangement and criteria modification are fixed. The $10X1$ family of restrictions requires that pooling and the arrangement of the pools are fixed while leaving the arrangement of the teams within the pools susceptible to manipulation. The $10X0$ family of restrictions requires a specific pooling but offers no other restriction on the seeding. It is not known if there are any competitions which use either the $011X$, $10X1$ or $10X0$ families of restrictions when generating seedings.

The $010X$ family of restrictions requires a specific arrangement of teams but requires no other restrictions. This model is used in scheduling the NCAA Division One Basketball Championship [48].

The most unconstrained family of restrictions is the $00XX$ family of restrictions. This is the model described in other papers [35, 28, 67, 70]. One of the more distressing results shown later is its wide use and relative susceptibility to manipulation. This type of completely unconstrained seeding is often used by local and amateur sports where it may be difficult to determine rankings and there may only be two pools. Also note that any competition where the seeding is generated by a random draw is seeded under the $00XX$ family of restrictions.

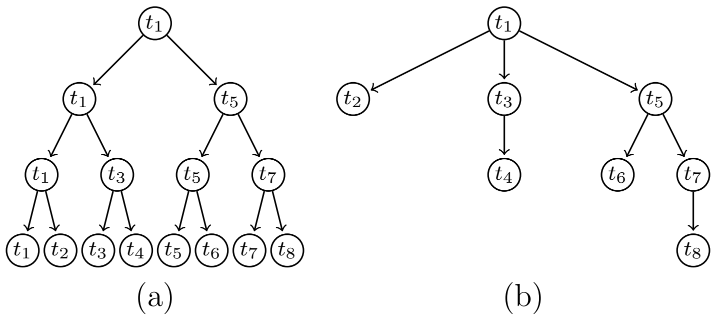


Figure 4.7: **(a)** A cup competition denoting the winner in each node. **(b)** The corresponding binomial spanning tree which represents the result of the competition.

4.4.3 Practical Complexity and Constraint Programming

While a more efficient method for solving these problems may exist, a polynomial-time algorithm for finding seeding manipulation strategies has yet to be discovered. The alternative to giving up on these problems would be to apply the machinery used to solve NP-Hard problems and look at the practical complexity of these problems. This constitutes neither a proof of efficiency or of hardness but rather shows, as is done in Graph Isomorphism [65], that these problems can be solved even if their complexity is indeterminate.

The basic problem is determining if a given team t_w is the winner under any seeding of the tournament given the tournament graph $G = (V, E)$. Lang et al. [35] showed that the least constrained problem is solved by finding a binomial spanning tree within the tournament graph structure. The winner of the cup represents the root of the tree. The children are the teams the winner defeated in each round from left to right. This notion is then applied recursively to each child. If there exists an assignment of the teams so that the edges in the binomial spanning tree are also in the tournament graph then the team at the root is a winner. In Section 4.4.4, it is discussed how these binomial trees must exist for all possible seedings.

Example 4.11 (Binomial Tree Winner)**.** *Figure 4.7a shows a cup competition with the expected winner of each match according to the tournament graph labelled in each node. The winner of the cup is t_1 and t_1 becomes the root of the binomial tree. Since t_1 defeats t_2, t_3 and t_5 in each successive round, t_2, t_3 and t_5 are made the children of t_1. t_2 does not win a single game so they are a leaf node. t_3 defeats t_4 and t_5 defeats t_6 and t_7. Finally, t_4 and t_6 are leaf nodes and t_7 defeats t_8, which is a leaf node. The completely binomial tree is shown in Figure 4.7b.*

To tackle these problems, a constraint programming framework for solving sub-graph isomorphism is used [72]. The constraint approach for the simple model given the notation

```
Algorithm:EdgeConstraints($E, v_1, \ldots, v_m$)
input  : A list of edges and a set of variables
output: Returns a set of constraints on the variables
c ← 1;
queue.add (pair ($v_c$,$log(m)$));
c ← c + 1;
while ¬queue.empty () do
    pair (current,num_child) = queue.pop ();
    for $i = 0$ to num_child − 1 do
        Add the constraint that the (current, $v_c$) be an edge in the tournament;
        // add a new pair to the queue
        queue.add (pair ($v_c$, $i$));
        c ← c + 1;
return the set of generated constraints;
```

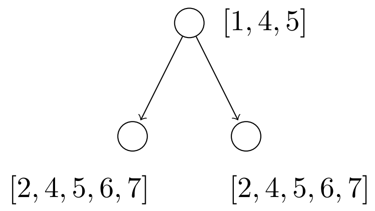
Algorithm 4.3: This algorithm uses a queue-based algorithm to generate the edge constraints for all variables in the constraint program.

is described first and then additional constraints are added to solve the other models. There are m variables labelled $v_1, \ldots v_m$, with the domain of each variable $D(v_i) = [1 \ldots m]$. Each variable, v_i is a node in the binomial spanning tree and the value of v_i is the corresponding tournament node. An AllDifferent constraint is added to ensure that no node is used more than once, *alldifferent*$(v_1, \ldots, v_m)$. Edge constraints are added to the variables of the graph to ensure that if there is an assignment of v_i and v_j to the variables in the spanning tree which share an edge then $(v_i, v_j) \in E$. Algorithm 4.3 describes how the edge constraints are generated for a given problem. Lastly, the constraint that $v_1 = t_w$ ensures that t_w wins the tournament, if possible.

Arc consistency is enforced on the AllDifferent constraint and the edge constraints. Using the constraint program described above and arc consistency, simulated realistic problems with hundreds of teams are solved in under a second.

This basic constraint-based approach can be extended with additional constraints to reflect the additional restrictions. The most useful is a global constraint proposed by Larrosa and Valiente [36], called nRF+ in that work but referred to here as the edge cardinality constraint, that enforces slightly stronger consistency on the children of nodes within the tree. This ensures that for each value of a parent node there are enough distinct values in the domains of the child nodes to satisfy each child node. Since, it is possible for both the AllDifferent constraint and the edge constraints to be arc consistent but there be no viable solution to the problem, this additional constraint can be effective.

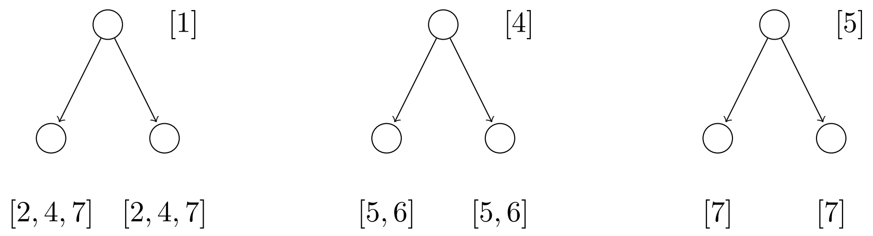
Example 4.12 (Edge Cardinality Constraint)**.** *Figure 4.8a shows a parent variable with*



(a)

	1	2	3	4	5	6	7	8
1	0	1	1	1	0	0	1	1
2	0	0	0	1	1	0	1	0
3	0	1	0	0	0	1	0	0
4	0	0	1	0	1	1	0	1
5	1	0	1	0	0	0	1	1
6	1	1	0	0	1	0	0	0
7	0	0	1	1	0	1	0	0
8	0	1	1	0	0	1	1	0

(b)



(c) (d) (e)

Figure 4.8: **(a)** An example of the edge cardinality constraint applied to a node with two children with the current domains of each node. **(b)** The tournament graph. **(c)** The result if the parent is set to 1 with three children. **(d)** The result if the parent is set to 4 with two children. **(e)** The result if the parent is set to 5 with only one value remaining so 5 can be pruned from the domain of the parent.

two child variables and Figure 4.8b shows the tournament graph associated with the problem. The domains of all three variables are arc consistent with respect to the edge constraints and the AllDifferent constraint and no further reductions can be made with those constraints. If an edge cardinality constraint is added that enforces that children which have the same parent must have between them enough values to assign each child then a further reduction can be made. Figures 4.8c and 4.8d show that if the parent is assigned the value 1 *or* 4 *then there are enough remaining values to assign to both children. However in Figure 4.8e, the parent is assigned the value* 5 *and a single value* 7 *remains in both children. Therefore, the value* 5 *can be removed from the domain of the parent.*

If there is pooling, a modified version of the constraints can be used so that the opponents in the early rounds are all from a given pool. This includes the edge constraints and the edge cardinality constraint. For edge constraints, it is sufficient to verify that for arcs representing early round matches that each child value has a possible parent value in the same pool. Similarly for edge cardinality constraints, only those values which represent

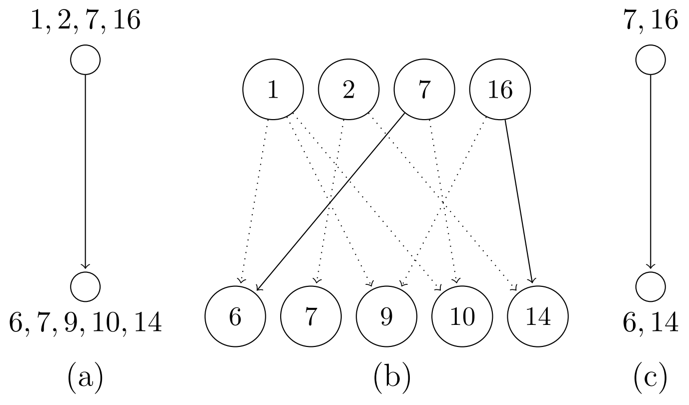


Figure 4.9: **(a)** A parent child pair of variables where the values are arc consistent according to the edge constraint. **(b)** The selection of the tournament graph corresponding to the values in the domains. Dotted lines are the pairings of values which do not respect the pooling restriction where the pools are $\{1 \dots 4\}$, $\{5 \dots 8\}$, $\{9 \dots 12\}$ and $\{13 \dots 16\}$. **(c)** The arc consistent domains of the variables after pruning once the pooling restriction is applied.

the same pool are counted. However, given the teams are fixed in the pools, it is possible to determine the set of possible pool winners from each pool as a sub-problem. It is often beneficial to take the set of pool winners and find a solution among the pool winners. Additional constraints must be added to ensure that exactly one pool winner is present in any solution.

Example 4.13. *Figure 4.9a shows an edge constraint which is arc consistent. The relevant sub-graph of the tournament graph can be viewed in Figure 4.9b. However, if teams must belong to the same pool then extra pruning can be achieved. Assume that teams are pooled into four groups* $\{1 \dots 4\}$, $\{5 \dots 8\}$, $\{9 \dots 12\}$ *and* $\{13 \dots 16\}$. *Therefore, the dotted edges in Figure 4.9b no longer provide support as the teams do not belong to the same pool and, as shown in Figure 4.9c, the parent values of* 1 *and* 2 *can be removed along with child values* 7, 9 *and* 10.

If there is a fixed team arrangement, the edge constraint can be modified so that every edge satisfies the arrangement. Suppose there is an edge between two vertices v_i and v_j. Assume that each team in v_i has a fixed rank and that the pair arrangement specifies which teams v_i could face, therefore, if there is a team $t_i \in D(v_i)$ then there must be a team $t_j \in D(v_j)$ such that (t_i, t_j) is an edge in the tournament graph and the rank of t_i is compatible with the rank of t_j given the round which the game is being played.

Example 4.14. *Figure 4.10a shows an edge constraint with arc consistent domains according to the relevant selection of the tournament graph shown in Figure 4.10b. Assume*

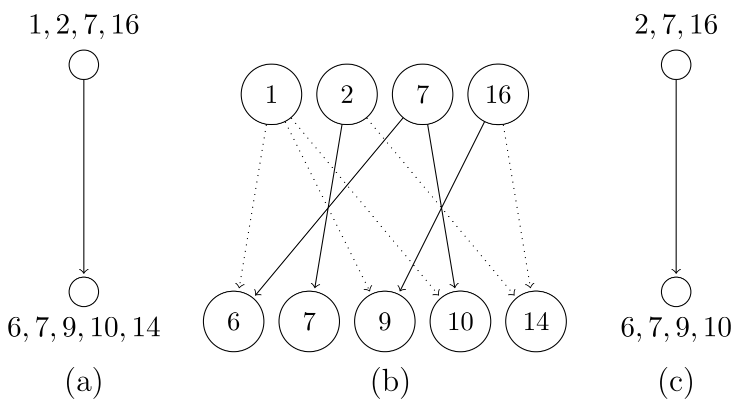


Figure 4.10: **(a)** A parent child pair of variables where the values are arc consistent according to the edge constraint. **(b)** Assuming that the edge in (a) represents a match in Round 1, the rank of a team i is $((i-1) \mod 4)+1$ and teams of rank 1 play rank 4 and rank 2 play rank 3, this figure shows the selection of the tournament graph where the edges that do not match the arrangement are dotted. **(c)** The arc consistent domain values after pruning using the fixed arrangement restriction is applied. Note that if pooling was added as in Example 4.13 and Figure 4.9, then the only values remaining would be 7 in the domain of the parent and 6 in the domain of the child.

that the edge represents a game in the first round and the arrangement criteria requires that top teams play the worst teams and so forth. In this example, assume teams are ranked $1,\ldots,4$ *and that the rank of a team* i *is equal to* $((i-1) \mod 4)+1$. *The dotted arcs in Figure 4.10b represent all of the edges where the match up would violate the arrangement. For example, if team 1 has a rank of 1 and therefore must play a team of rank 4 but the opponents of team 1 remaining according to the original domains are 6, 9 and 10 which have rank 2, 1, and 2, respectively. Therefore, according to the arrangement, team 1 has no valid opponents and can be removed from the domain of the parent variable. Figure 4.10c shows the domains of the variables after all reductions have been made.*

In a similar manner to modifying the edge constraint to deal with pair arrangement, if there is a fixed arrangement of pool winners then this can be enforced by ensuring that the edge is in the tournament graph and it satisfies the arrangement.

When the criteria by which an arrangement is made, for example the rank, is fixed then determining which teams satisfy the arrangement is straight forward. However, if the criteria is allowed to be modified slightly then an additional variable is added for each node in the binomial tree. This variable represents the value of the criteria for the team at the given node of the tree. Additional constraints are added to ensure the criteria matches with the team such as, for example, that the rank is not too different from the actual rank.

The edge constraint is modified again to ensure the possible values of the criteria allow for a given team arrangement.

4.4.4 Symmetry Removal

The simplest representation of a seeding is the simple list representation where the i^{th} element in the list represents the i^{th} leaf of the tree. One of the problems with a list representation is that it contains many rotational symmetries.

Example 4.15. *Suppose there existed a sixteen team cup with the seeding represented as* $\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16\}$*. In this example, in the first round, team one plays team two, team three plays team four and so forth. Note however, that* $\{9, 10, 11, 12, 13, 14, 15, 16, 1, 2, 3, 4, 5, 6, 7, 8\}$ *and* $\{8, 7, 6, 5, 4, 3, 2, 1, 9, 10, 11, 12, 13, 14, 15, 16\}$ *are identical seedings. The match ups in the first round stay the same, team one plays team two, team three plays team four and so forth. Given the list representation, it is possible to rotate the position of every leaf game and every sub-tree of games without disturbing the result of the competition as the match-ups remain the same. Given that there are* $m - 1$ *games played in the tournament where each of the games could be independently rotated, there are exactly* 2^{m-1} *equivalent seedings for every seeding represented as list.*

Another possible representation is the binomial tree representation proposed by Lang et al. [35]. Lemma 4.8 shows that every seeding represented as a list can be transformed into a binomial tree, given the tournament graph.

Lemma 4.8. *Given a tournament graph, a set of teams (where* $m = 2^n$*) and a seeding, there always exists a binomial tree structure that can be constructed from the seeding where for every edge in the binomial tree there is an edge in the tournament graph.*

Proof. First note that, using the tournament graph, the winner of every game in the cup can be determined given the seeding of games. Now, the binomial tree is constructed recursively. The winner of the competition t_w is placed at the root of the binomial tree. Next place all of the teams that lost directly to that t_w in the leaves from left to right in the order that t_w beat the teams, which is exactly $log(m)$ games. Now, recursively add the children of each node ensuring that the children are added from left to right in the order the children were defeated. The tree is constructed so that a parent node would have played their i^{th} child node in the i^{th} round of the tournament. Therefore for the i^{th} child to have lost in the i^{th} round, they must have won exactly $i - 1$ games. This means that, given the children of a parent node who wins k games, exactly one child has 0 children, one child has 1, and so forth, with the last child having $k - 1$ children. This is the definition of a binomial tree. □

The binomial tree only specifies who plays each team in each round but not to which leaf the team belongs as any rotation can be used with identical results. Therefore, using the binomial tree representation preserves all solutions without having to search through the exponential number of rotational symmetries.

4.4.5 Experimental Results

While there is some evidence to suggest there is bias in seeding real world cup competitions [17], there is no definitive data set to test the efficiency of the constraint programming approach described above. Therefore to test the constraint programming approach under the various restrictions, a set of realistic test benchmarks are generated. The first step is to generate a realistic tournament graph.

To generate a real world tournament graph, real world data was mined to generate a probability distribution of how often better teams lost to worse teams and their relative difference in terms of rank. The NCAA Division One Basketball competition brackets from 1985 to 2009 were mined for this information. A probability model was constructed from the data to give the distribution of upsets given the difference in rank. This model can be seen in Figure 4.11. The graphs relative unevenness at high differences in rank is likely due to there being very few instances for that difference. For example, for a difference of 10 rank, there is a forty percent chance of upset, which is likely due to the fact that there was only 5 times where a team ten ranks apart played and 2 of them ended in an upset.

Tournament graphs are generated by sampling the distribution for each game to determine whether the lower ranked team defeated the higher ranked team. Graphs of size 4, 8, 16, 32, 64, 128 and 256 were generated. The largest cup balanced competitions found in practice are those used by professional tennis where there are 128 participants. For each tournament graph, every team is tested to determine whether there exists a seeding which would ensure their victory. All of the families of restrictions were tested except the 1110 family, since it is a strictly easier variant of the 0000 family, and the 1111 family, which can be solved in linear time using simple checks. Table 4.3 shows three of the more interesting families of restriction. The timing results show that for most families of restrictions the random instances generated are quickly solved. Only in a single family of restrictions, the $010X$ family, was there problems which timed out and even then there were relatively few of those. Complete results can be found in Table B in the appendix. Table 4.4 shows the percentage of teams that could be manipulated for a given rank. The results from Table 4.4 confirms the theoretical result on the likelihood of possible seeding manipulation strategies on the most unrestrictive model [70]. However, even relatively small restrictions can drastically reduce the number of teams which could be made winners via seeding manipulation. As expected from the generation of the tournament graph, it is more likely that the scheduler could generate a seeding manipulation strategy for strong teams. The

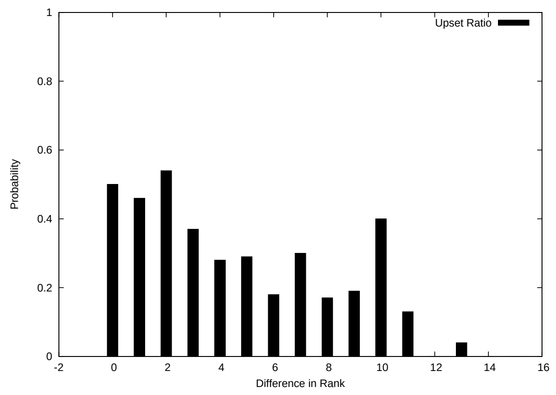


Figure 4.11: The probability that a team whose rank is i positions lower than another team would upset that team as estimated from the results for the 25 tournaments from 1985 to 2009. For each game in every tournament, the distance in rank between the two teams was calculated and it was determined if the weaker team upset the stronger team.

complete table can be found in Appendix B in Table B.2. The results show that, at least on random instances, that seedings can be easily generated to manipulate the results of the tournament. Therefore, the cup competitions are open for potential abuse by the scheduler of the competition from a practical point of view.

4.5 Combining Manipulations

In this section, the combination of the three manipulation schemes is discussed. Specifically, how the problem changes if a scheduler has a coalition of cheaters with which to change the outcome of games and how difficult it is to manipulate games in the two-stage competitions where the first stage is a round robin and the second stage is a cup competition.

4.5.1 Seeding Manipulation with a Coalition of Cheaters

In Section 4.4, it was shown that, given the random model studied, it is almost always easy to manipulate a cup competition. However, it is not always possible for the scheduler to manipulate the results especially for lowly ranked teams or as restrictions are added to the problem. The question becomes how does the problem change if the scheduler has access to a coalition of willing cheaters that could throw games as needed.

Table 4.3: Effect of tournament size on the minimum, maximum and average CPU time (sec.) to calculate a seeding manipulation, for selected families of restrictions $00XX$, $10X0$ and $010X$. The complete results for all families are shown in Appendix B in Table B.

	00XX		10X0		010X	
size	range	avg	range	avg	range	avg
16	[0–1]	0.00	[0–1]	0.00	[0–1]	0.01
32	[0–1]	0.01	[0–1]	0.03	[0–1]	0.04
64	[0–1]	0.04	[0–1]	0.06	[0–1657]	1.3
128	[0–1]	0.27	[0–1]	0.13	[0–?][1]	N/A[1]
256	[1–61]	1.81	[0–1]	0.33	[0–?][2]	N/A[2]

[1] Due to memory restrictions, it was not possible to solve 9 out of the 12800 instances. As such, the maximum time and average cannot be accurately reported.

[2] Due to memory restrictions, it was not possible to solve 1794 out of the 25600 instances. As such, the maximum time and average cannot be accurately reported.

Another way of looking at this problem is to notice that the tournament graph has a single edge between any two pairs of teams i and j. When a game is thrown, the reverse of the edge occurs. So any game where the coalition member is expected to win, the reverse edge can be added to the tournament graph. With these modifications to the tournament graph, all of the constraint programming techniques used in Section 4.4 can now be applied to the new problem.

To test that the problems remain easy, random coalitions were generated and tested on the tournament graphs created in Section 4.4.5. Coalitions have at least one member and contain at most half of the teams. The size was selected from this range using a uniform random distribution. The teams making up the coalition were also generated randomly. To compare the differences, the solver for the $00XX$ restrictions was applied to the tournaments with and without coalitions. Table 4.5 shows there is no practical difference in solving the time taken to solve instances where coalition members are added and Table 4.6 shows that there are strictly more possible solutions given the effect of the coalition.

4.5.2 Combining Round Robin and Cup Competitions

There are a variety of two-stage competitions that are available but the most common form of two-stage competition involves a group stage followed by a fixed cup competition. In the following section, several instances are described where there exists a polynomial-time

Table 4.4: The effect of tournament size on the percentage of teams of rank i that could be made the winner via manipulation of the seeding, for selected families of restriction: $00XX$, $10X0$, and $010X$. The complete results for all families is shown in Appendix B in Table B.2.

00xx	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
4	41	41	41	41	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
8	91	80	80	85	63	54	43	40	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
16	97	97	96	95	95	94	95	95	90	92	90	82	78	62	59	32
32	100	100	100	100	100	100	100	100	100	100	100	100	100	98	97	87
64	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
128	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
256	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
10x0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
16	97	97	96	95	95	94	95	95	90	92	90	82	78	62	59	32
32	97	96	95	96	94	93	94	93	91	89	87	85	76	67	48	37
64	99	98	99	99	98	98	98	98	95	96	92	88	78	64	52	40
128	99	97	97	97	97	96	96	96	94	92	91	85	75	66	55	44
256	98	98	97	97	97	97	96	96	94	93	90	87	77	65	56	39
010x	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
16	88	93	79	68	58	51	35	20	12	22	13	11	7	1	1	0
32	100	100	100	100	100	100	100	93	89	99	93	90	79	69	23	6
64	100	100	100	100	100	100	100	100	100	100	100	100	100	97	54	16
128	100	100	100	100	100	100	100	100	100	100	100	100	100	100	75[1]	32[1]
256	100	100	100	100	100	100	100	100	97[2]	100	100	100	96[2]	91[2]	52[2]	0[2]

[1] Percentages are a lower bound of actual percentages with a difference from actual between 0% and 1%.

[2] Percentages are a lower bound of actual percentages with a difference from actual between 0% and 51%.

decision method for determining a constructive manipulation in this type of competition. For this section, it is assumed that the scoring model is the simple win-loss model.

The simplest case of combining these two problems is when there is a round robin for each single place in the cup competition. Determining a manipulation in this case can be done in polynomial time.

Theorem 4.7. *A constructive manipulation in a two-stage competition where the winner of the groups qualifies for a set spot in a fixed cup can be determined in polynomial time.*

Proof. First note that regardless of the size of the round robin groups, determining the winner under manipulation can be done in polynomial time (for the win-loss model, see Section 4.3). Therefore, it is possible to generate a set of possible winners at each fixed

Table 4.5: The effect of tournament size on the minimum, maximum and average CPU time (sec.) for determining whether a team could be manipulated by seeding manipulation for the $00XX$ family of restrictions with or without a coalition.

	Without Coalition		With Coalition	
size	range	avg	range	avg
4	[0–0]	0.00	[0–0]	0.00
8	[0–0]	0.00	[0–0]	0.00
16	[0–1]	0.00	[0–1]	0.00
32	[0–1]	0.01	[0–1]	0.01
64	[0–1]	0.04	[0–1]	0.03
128	[0–1]	0.27	[0–1]	0.26
256	[1–61]	1.81	[1–10]	1.97

Table 4.6: The effect of tournament size on the percentage of teams of rank i that could be made the winner via manipulation of the seeding for the $00XX$ family of restrictions with and without coalitions.

No Coalition	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
4	41	41	41	41	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
8	91	80	80	85	63	54	43	40	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
16	97	97	96	95	95	94	95	95	90	92	90	82	78	62	59	32
32	100	100	100	100	100	100	100	100	100	100	100	100	100	98	97	87
64	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
128	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
256	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
Coalition	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
4	67	63	67	72	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
8	97	91	95	92	85	82	78	79	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
16	97	98	97	96	96	96	96	96	95	96	95	96	96	89	92	89
32	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	99
64	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
128	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
256	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100

leaf in the cup. Given a fixed partition of teams at the leaves, the modified CSL algorithm described in Section 4.2 can be applied where at round one the sets must be compared instead of single teams. The remainder of the levels can be calculated as before. Since both parts can be calculated in polynomial time regardless of how many teams are in the group pools and the cup, the entire solution can be calculated in polynomial time. □

Another common method is to have the top two teams from each group qualify into two predetermined spots in the cup. This is the situation that is most often used in tournaments like the World Cup of Football. In the World Cup, teams from the same group are placed on opposite sides of the cup competition and would not face each other again until, if both kept winning, the final. Since there are two teams from each group which both may be able to finish in first or second, the possible winners at the leaves may contain duplicates and, therefore, do not form a partition. Given the lack of a partition, the complexity of this problem is unknown in general but, interestingly, it is polynomial to determine if a team makes the finals. First, a lemma used to prove the theorem,

Lemma 4.9. *There is a polynomial time algorithm for determining the set of teams that can finish in first and second place in a round robin tournament.*

Proof. While it is NP-Complete to determine if a team finishes in k^{th} place, in this case the problem is to determine all teams that could finish in second place or better. It is easy to determine if a team can finish first given the win-loss scoring model. Determining if a a team can finish in second is harder but, given there is at most $O(n)$ teams in a pool, it is possible to determine whether each team in the pool is better than all but one of the other teams. Each team i is paired with every other team j and the maximum number of wins that each can get while $w_j \geq w_i$ is determined. This can be done by finding the maximum number of wins possible for i and j not using the games against each other. Next, the wins to between the two teams are added, iteratively, to the team with the fewest wins, the minimum win total between the two teams is used as the target point value. Using the feasible flow method described in Section 4.3 with a t_i needing exactly the target and t_j needing equal or more than the target, a solution means that there exists a situation where t_i finishes second. Hence there are $O(n^2)$ ways of configuring i and j and each can be solved in polynomial time using feasible flows. □

Theorem 4.8. *Determining if a coalition S can manipulate a two-stage competition, where teams from the same pool are placed on opposite sides of the cup competition, so that a team t_w reaches the finals can be done in polynomial time.*

Proof. Using Lemma 4.9, it is possible to determine in polynomial time the list of teams that finish in first and second in each round robin group. The second stage now has a set of teams filling the correct positions of the cup and, given the structure of the cup, sets that contain the same team are on opposite sides of the cup. Since the problem is to determine whether there exists a way for the coalition to make a team t_w a team in the finals, it suffices only to look at half of the cup competition at a time. Therefore, the modified CSL algorithm is applied replacing the single team at the leaves with the set of possible teams. Since there are no duplicates and extra conditions, this can be solved in polynomial time. Therefore, since both stages can be solved in polynomial time, it is possible to manipulate a team so that it makes the finals when teams in the same group pool are seeded on different sides of the competition. □

Theorem 4.8 works only in the case of World Cup style competitions where exactly two members of each group qualify for the finals. The number of ways to seed the top k teams grows exponentially and, while these problems may still be solvable as in Chapter 3, the problem no longer remains polynomial. The general problem where there are k teams from each group making the finals is NP-Complete as the first stage is NP-Complete [45].

Given the result that all pairings of first and second place teams can be found in polynomial time, it is possible to look at winning the tournament instead of just reaching the final. The problem is complicated by the fact that some teams can finish in either place and not every first and second place option can happen simultaneously. Due to this fact, the complexity of the general problem remains open but there are two variants of the problem that are polynomial. The first variant is found by noting that much of the hardness comes from the fact that teams from the same pool are placed on opposite sides of the cup competition. For example, if the teams were required to playoff for the same leaf of the competition, then it is possible to construct the list of possible leaves from each pool which can be solved in polynomial time. The example is generalized slightly in the following theorem.

Theorem 4.9. *A polynomial algorithm exists for determining constructive manipulations for two-stage competitions where two teams are seeded as in the World Cup but they are within constant distance of each other in the tree.*

Proof. Given Lemma 4.9, there is a polynomial time algorithm for determining the pairs of teams that finish in first and second. Note that if teams are a constant distance apart in the tree then there is a constant sized sub-tree containing both pairs of teams and a set of other pairs which are of equal distance apart. Therefore, all of the possible winners of that sub-tree without conflicts can be determined in polynomial time by enumerating all possibilities. Once there is a set of valid non-conflicting winners of each sub-tree, the modified CSL algorithm with the set of possible winners at the leaves as in Theorem 4.7 can be used to calculate the remaining possible sets of winners in polynomial time. □

The second variant uses the same idea as in Sections 4.2.2 and 4.2.3 that if there are only a few coalition members then they can only have a local impact on the cup competition. Again, this idea relies on the assumption that there is likely to be only a few manipulators in a competition and, from a practical perspective, this method could be applied.

Theorem 4.10. *A polynomial algorithm exists for determining constructive manipulations for two-stage competitions where two teams are seeded as in the World Cup but there is a coalition of constant size c.*

Proof. Given at most c coalition members, that most c of the groups have a manipulator. Given this, at most a constant number of groups could have different choices in selecting the first and second place teams. Since there are at most $O(m)$ teams in each pool, there

Table 4.7: The complexity of the various different manipulation strategies discussed in this chapter.

Manipulation Type	Worst-Case Complexity
Constructive Cup	Polynomial
Destructive Cup	Polynomial
Minimal Cup	Polynomial
Double Elimination Cup	Polynomial for a bounded sized coalitions
Reseeding Cup	Polynomial for a bounded sized coalitions
Constructive Round Robin	Polynomial for certain scoring models, NP-Complete otherwise
Destructive Round Robin	Polynomial
Seeding	Open
Combining Cup and Seeding	Open
Combining Cup and Round Robin	Polynomial under conditions

are at most $O(m^2)$ possible winners from a pool and $O(m^{2c})$ different options for seeding the tournament which is polynomial. Since each option can be checked in polynomial time by the modified CSL algorithm, the solution can be found in polynomial time. □

4.6 Summary

In sporting tournaments, unlike elections, manipulations are applied directly to the tournament graph. In this chapter, three different types of manipulations have been discussed that directly manipulate the tournament graph. As well, combinations of the three types of manipulations are discussed. Algorithms used in elections and winner determination problems were extended to the context of sporting manipulations.

Cup manipulations are easily manipulated and it is shown that modifying the CSL algorithm used to find manipulations in elections gives an efficient method for calculating the manipulations in tournaments. It turns out that finding the minimal number of manipulations, and thus the minimal sized coalition, is also easy in cup competitions. Some variants of cups, specifically double elimination cups and reseeding cups, can be solved in polynomial time if the size of the coalition is of bounded size.

Round robin competitions are also examined and it is shown that, for a wide variety of scoring models, manipulations can be determined in polynomial time. This can be observed by noting that there is an equivalence between the winner determination problem and the round robin manipulation problem. It is shown that calculating the minimal number of manipulations in round robins is also polynomial.

While the complexity of seeding a tournament remains open, the practical complexity of the problem was studied by modelling the problem as a constraint program. Without restrictions, on randomly generated instances, the problem remains easy and solutions are found quite fast even on instances with up to 256 teams. This holds true with many of the restricted models and, even on the most difficult model, the problems are mostly very easy.

Combining the different manipulations provides some interesting results. Combining manipulation of the seeding along with manipulation of the games in the cup has unknown complexity but is again solved easily in practice since the solution requires only adding extra arcs to the tournament graph. The complexity of manipulating a combination of a round robin and a cup competition used in the World Cup of Football, Olympic Hockey and a host of other sporting events is polynomial-time solvable under certain conditions. If a single team qualifies from each group, there exists a polynomial time algorithm for determining a manipulation strategy. If two teams qualify from the group, determining if the team reaches the final can be calculated in polynomial time but the problem of determining the winner is of unknown complexity. If restrictions are put on where the teams from each group are seeded or how many coalition members exist in the tournament, then the problem can again be solved in polynomial time.

In the next chapter, manipulation is examined from another perspective. Instead of determining how hard it is to manipulate a competition, the detectability of those manipulations is examined.

Chapter 5

Detecting Manipulation in Sporting Tournaments

The focus of research on manipulation in elections and sports has been on how difficult it would be for the cheaters to achieve their ends. There has been very little focus on determining whether the cheating could be detected by the organizers of the tournament. One reason why this may have been a neglected area of research is that most known examples of match-fixing are single events where the cheaters hope to profit from betting [5]. However, all of the previous work on manipulation deals with global manipulation strategies where coalitions change their behaviour to modify the winner of the tournament. Since these type of manipulations affect the tournament on a broad scale, it raises the question of whether it is possible to detect such manipulations.

This chapter focuses on the detection of the three different types of manipulations described in Chapter 4: cup manipulation, round robin manipulation and seeding manipulation. Detection of manipulation can be broadly separated into three different types: detection based on events occurring within the individual games, detection based on events external to the competition like betting and detection based on a pattern of game results. Corresponding to the manipulation results from Chapter 4, this chapter focuses on the third type, detection of manipulation based on a pattern of game results. As such, when a certain manipulation strategy is labelled as undetectable, it is meant that there exists no difference in the pattern of game results from any fair results. Conversely, a manipulation strategy is labelled detectable if the pattern of game results exhibits optimality—uses as few changes to the expected results as possible—while fair behaviour that achieves the same result does not necessarily do so.

In Section 5.2, the detection of cup manipulations is examined and the concept of a strategically optimal coalition is introduced. It is shown that strategically optimal behaviour occurs rarely in randomly generated tournaments and can be used to provide detection assistance. This type of detection must be paired with individual game results as a small percentage of teams can be misclassified and individual game analysis is proposed as a complementary technique to detect manipulation. In Section 5.3, the detection of round robin manipulation is proposed and two different classes of manipulation are examined. Some simple algorithms are proposed along with a discussion of the results in terms of detectability. In Section 5.4, several families of results are identified as being undetectable based on the result of the seeding. The criteria modification restriction parameter is shown to be the crucial factor in detectability as it is possible to identify the difference from the expected results whereas with all other unfixed parameters any seeding would be possible given a random draw.

5.1 Related Work

While there is no work on detecting coalitions of cheaters in election results, there is some work on determining if the results of elections were fair within some confidence interval. This process is known as auditing [7]. Aslam et al. [7] present an auditing technique that is robust even when the electoral precincts are of different sizes. Myagkov et al. [47] and Levin et al. [38] study election fraud using statistical auditing for elections in Russia, Ukraine and Venezuela. The work in this chapter attempts to detect the same type of behaviour, i.e., coalitions of cheaters attempting to change the result, but using different techniques and focusing on a different aspect of the results.

In game theory, there is the notion of a coalitional game [50]. These are games where the goal is to find the best partitioning of agents such that some metric is maximized, for example, social welfare. Rahwan et al. [53] propose a mechanism for making anytime decisions for producing coalitions that gives the best result. The work in coalitional games is similar as coalitions are found using some fitness measurement. The reason for finding the coalition differs in this work as coalitional behaviour is not desired.

The area of network security also contains work that is related to coalition and cheating detection. Yan [71] proposes some methods to deal with illicit play of coalitions of players in online bridge. Brickell and Stinson [12] propose a threshold scheme for protecting encrypted data that allows cheaters to be detected with high probability even when working in a coalition. Jin et al. [32] look at finding coalitions of cheaters attempting to circumvent copyright procedures on streamed content. The work in coalition and cheating detection in network security is focused on specific technical problems and cannot be directly applied to the work in this chapter.

5.2 Detecting Manipulations in Cup Competitions

In cup competitions, it is known that a coalition can generate a manipulation strategy in polynomial time (see Chapter 4). Combining this fact with the observation that in many real world tournaments a large percentage of games are considered upsets by the organizers (see Figure 5.7 in Section 5.2.4), it poses the question of whether or not some of those upsets are strategic choices made by a coalition of cheaters.

In this section, a practical method for determining whether the behaviour of a coalition of teams in the tournament matches the behaviour of a manipulating coalition is described. The practical benefit of this technique would be to provide the organizers of cup competitions with a tool to identify when suspicious patterns of behaviour have occurred. Note that this technique does not provide a proof of dishonest behaviour but rather provides organizers with a significant starting point to investigate possible corruption in the competition.

It is well known that manipulations occurs in sports. This includes sports ranging from soccer [25, 5], baseball [30], tennis [23], sumo wrestling [21] and even lawn bowling [14]. Detection of these events is often done through an analysis of betting and financial records [25, 5, 23] but cheating is also apparent in the pattern of results of the game. The challenge becomes disentangling the legitimate upsets from the manipulations. For a single game, it may be possible to identify a manipulation via analysis of betting results or on field play. However, for a sequence of games or a tournament of games, it becomes possible to highlight suspicious patterns of activity rather than isolated occurrences. The remainder of this section focuses on providing a technique for identifying suspicious patterns of behaviour within the tournament results. The resulting algorithm could be used to aid investigators and target specific teams or groups of teams whose on field play had yielded suspiciously coincidental play.

In order to provide a foundation for detecting tournament manipulation, the concept of a strategically optimal coalition is introduced along with a discussion about its relevance in detecting fraud in sports. Some restrictions and observations about strategically optimal coalitions are described and a dynamic programming approach is presented which easily solves examples which are twice as large as known cup competitions. While the complexity of the problem remains unknown, empirical results show that detecting coalitions in large tournaments is possible.

5.2.1 Notation and Definitions

Before introducing the concept of strategically optimal coalitions, some notation and definitions are introduced formally. Refer to Section 2.5 for more details on the notation and concepts used in this chapter. Since it is much harder to determine cheating before the competition has completed, it is assumed that the results of the cup competition are given and the winner of every match is known. The final winner of the competition is denoted t_w. It is also assumed that it is possible to identify accurately an upset from an expected victory. The concept of a possible upset relaxes the assumption that the tournament graph perfectly predicts the outcome of games. However, it is possible that some games are identified as an upset or an expected outcome when the reality is the opposite.

At each round $i = 1, \ldots, n$ of the competition, if t_w wins the tournament as a result of manipulation by a coalition of teams, the coalition must have formulated a strategy to guarantee that t_w wins. The strategy may need to change from one round to the next as the outcomes of many of the games will not be under the coalition's control and there may be upsets caused by teams that are not in the coalition. It is assumed in this work that rounds happen simultaneously and planning happens only before or after rounds. In general, this is restriction could be relaxed by simply adding possible plan changes between every game that did not happen simultaneously. As well, it is assumed there is a strong incentive for a coalition to minimize the number of manipulated matches, as each manipulation increases the risk of detection. The reason why this assumption is likely to be true is that additional unnecessary upsets are likely to draw attention to the coalition while providing no benefit especially since manipulating eliminates the coalition member.

Definition 5.1 (optimal manipulation strategy)**.** *Given a coalition S, a distinguished team t_w, and the results of past rounds $1, \ldots, k-1$ of the competition, a* manipulation strategy *for S in round k is a set of manipulations that if executed ensures t_w wins the tournament under the assumption that the matches between teams not involving teams in S occur as expected in the tournament graph G. A manipulation strategy for S in round k is* optimal *if there exists no strategy for S with fewer total manipulations.*

Example 5.1 (An Optimal Strategy)**.** *Consider a sixteen team cup competition where the tournament graph is as shown in Table 5.1. If the outcomes of the matches follow the tournament graph t_9 will win the cup (see Figure 5.1). Suppose that the coalition of t_{13} and t_{16} would like t_1 to win the cup. An optimal manipulation strategy for $S = \{t_{13}, t_{16}\}$ in round 1 is given by,*

t_{13} throws match to t_{14} in round 1,
t_{16} throws match to t_{14} in round 2.

Given that t_{13} then throws the match to t_{14} in round 1 and no other upsets occur in that round, an optimal manipulation strategy for round 2 involves just advancing the plan. If t_{16} then throws the match to t_{14} in round 2 and no further upsets occur in the remainder of the competition, t_1 wins the cup (see Figure 5.2).

The goal of this work is to recognize coalitions which may have manipulated the competition to have their team win. It is not possible to know the coalitions or their manipulation strategies but it is sometimes possible to recognize such coalitions through partial observation of their strategic behavior.

Definition 5.2 (strategically optimal coalition)**.** *A coalition S is a* strategic coalition *if for each round k, $1 \leq k \leq n$, the set of upsets U_S^k contains all and only the manipulations that would have been executed in round k in an optimal manipulation strategy for S in that round. A coalition S is a* strategically optimal coalition *if no proper subset of S is a strategic coalition.*

In the remainder of this section, coalitions are assumed to be formed prior to the competition. While this is a strong assumption, there are real world scenarios where this assumption would hold. The first scenario presumes that the financial activity which most likely accompanies such coalition formation is probably more closely watched while the competition is being played. Therefore, adding new coalition members on the fly exposes the coalition to increased risk. The second scenario would be where the teams spend their budget prior to the competition being played. In this scenario, no further coalition member could be added. A third scenario would be where the cost of adding the additional coalition member is more than the gain from winning the tournament. In these scenarios, adding extra coalition members would provide little benefit.

Another assumption is that real coalitions operate in a strategically optimal manner. It is possible that a coalition does not in fact operate such that they manipulate as few games as possible and contain only the necessary members. There are two arguments to why this assumption may be reasonable. The first is that believing that a coalition is not smart enough to operate seems dangerous and that a smart coalition would prefer to operate in a manner that manipulates as few games as possible and draws the least attention. The second argument is that a larger than necessary coalition would reduce whatever reward each coalition member receives.

The most restrictive assumption that this work makes is that it is assumed that if a coalition member is involved in an upset then the upset was a manipulation. While this is obviously a strong assumption, there is some reason to believe that if this is not true then the coalition would fail to manipulate the competition. Since this procedure is attempting to identify coalitions that successfully manipulated a coalition, it seems likely that the upset of a member would cause the coalition to be unsuccessful and, therefore, the method would not be directly applicable.

The relaxation of these assumptions is left for future work.

Example 5.2 (Changing a Strategy)**.** *Consider again the cup competition introduced in Example 5.1, where the coalition $S = \{t_{13}, t_{16}\}$ would like t_1 to win the cup. The optimal*

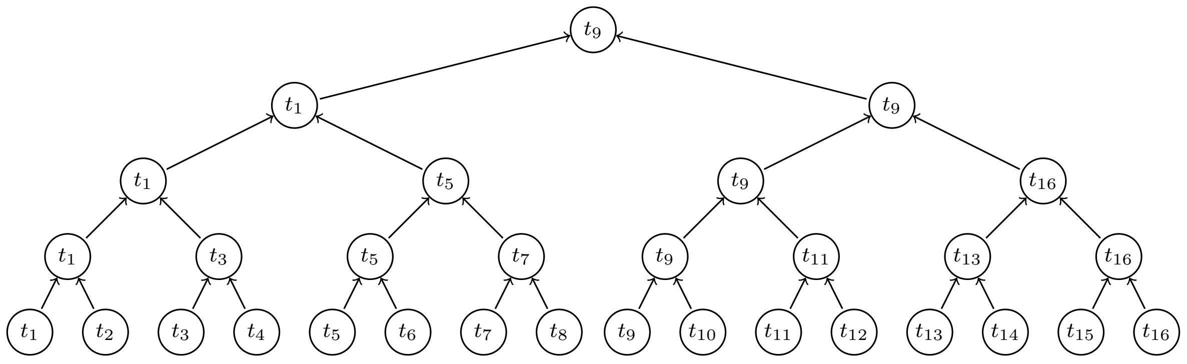


Figure 5.1: The result of the cup if matches happen according to the tournament graph.

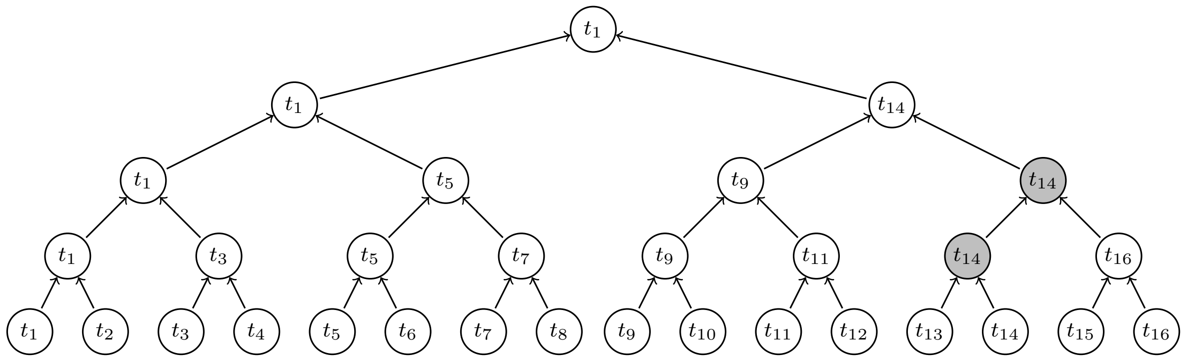


Figure 5.2: The result of the cup if t_{13} and t_{16} manipulate the tournament and all other matches happen as expected. Grey nodes indicate upsets.

manipulation strategy for S in round 1 remains the same and suppose that t_{13} then throws the match to t_{14} but that there are also additional upsets in round 1 involving teams outside of the coalition (see Figure 5.3). As their original strategy no longer ensures that t_1 wins the cup, the coalition must now reformulate their strategy by having t_{16} beat t_{14} and then ultimately lose to t_1. The strategy for round 2 is given by,

t_{16} throws match to t_1 in round 4.

Assuming no further unexpected upsets, the strategy carries over to subsequent rounds and t_1 wins the cup.

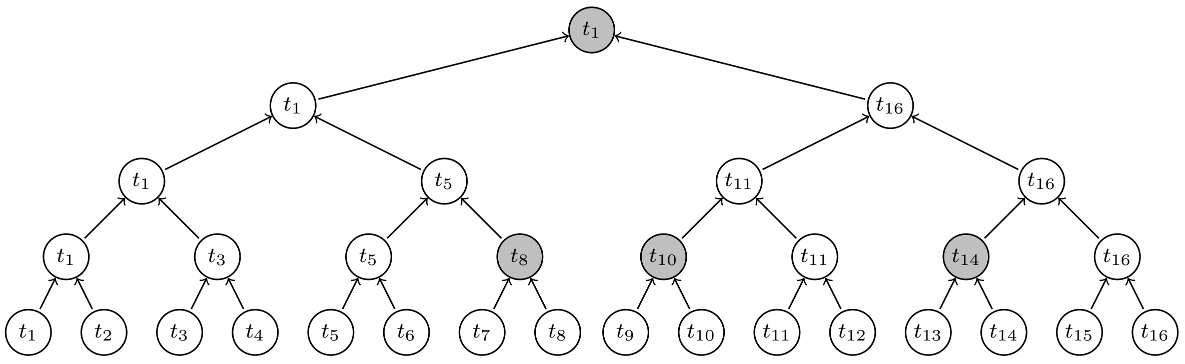


Figure 5.3: The result of the cup if t_{13} and t_{16} work in a strategically optimal manner and there are additional upsets caused by non-coalition members. Grey nodes indicate upsets.

Table 5.1: Adjacency matrix of the tournament graph for Example 5.1. There is an edge from t_i to t_j if cell (i, j) is equal to 1.

	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}	t_{11}	t_{12}	t_{13}	t_{14}	t_{15}	t_{16}
t_1	–	1	1	1	1	1	1	1	0	0	0	0	0	1	0	0
t_2	0	–	0	1	1	1	1	1	1	1	1	1	1	1	1	1
t_3	0	1	–	1	1	1	1	1	1	1	1	1	1	1	1	1
t_4	0	0	0	–	1	1	1	1	1	1	1	1	1	1	1	1
t_5	0	0	0	0	–	1	1	1	1	1	1	1	1	1	1	1
t_6	0	0	0	0	0	–	1	1	1	1	1	1	1	1	1	1
t_7	0	0	0	0	0	0	–	1	1	1	1	1	1	1	1	1
t_8	0	0	0	0	0	0	0	–	1	1	1	1	1	1	1	1
t_9	1	0	0	0	0	0	0	0	–	1	1	1	1	0	1	1
t_{10}	1	0	0	0	0	0	0	0	0	–	0	0	1	1	1	1
t_{11}	1	0	0	0	0	0	0	0	0	1	–	1	1	1	1	0
t_{12}	1	0	0	0	0	0	0	0	0	1	0	–	1	1	1	1
t_{13}	1	0	0	0	0	0	0	0	0	0	0	0	–	1	0	0
t_{14}	0	0	0	0	0	0	0	0	1	0	0	0	0	–	0	0
t_{15}	1	0	0	0	0	0	0	0	0	0	0	0	1	1	–	0
t_{16}	1	0	0	0	0	0	0	0	0	0	1	0	1	1	1	–

5.2.2 Pruning and Bounding the Coalitions

While there are potentially many different coalitions that could form, it can be relatively easy to prune teams from strategically optimal coalitions. In this section, a variety of techniques are discussed that can be used to show that certain teams could not be a part of a strategically optimal coalition which improves the efficiency of the dynamic programming algorithm presented in Section 5.2.3. The first pruning mechanism that can be applied uses the fact that a strategically optimal coalition must control the outcome of every game involving t_w, the team the coalition wants to win.

Lemma 5.1. *Any upset involving t_w has to be a manipulation and the team that causes an upset in a match against t_w must belong to every strategically optimal coalition.*

Proof. Assume that the upset involving t_w was not a manipulation. Therefore, the expected outcome was for t_w to lose and a strategically optimal coalition would not be possible as the coalition would not have been able to guarantee the result given the expected outcome. Hence, the team that causes the upset must be a member of every strategically optimal coalition. □

It can be observed that each problem can be decomposed into $\log(m)$ separate subproblems as shown in Figure 5.4, where m is the number of teams in the competition.

Lemma 5.2. *If a coalition S is manipulating to make t_w the winner, then this problem can be solved as $\log(m)$ problems of arranging the opponents of t_w such that in every round k, $1 \leq k \leq n$, the opponent either loses to t_w or is a coalition member that manipulates the game so t_w wins.*

Proof. If the team t_i facing t_w is not a coalition member and t_w loses to t_i according to the tournament, then the coalition has not achieved its goal as t_w would be eliminated and would not win the competition. To prove decomposability, it must be shown that changing the opponent of t_w in a round r_i can be done in isolation from finding the opponent in any other round. Observe that teams can only affect each other if they could play each other. Since teams always are eliminated after losing, for these teams to face each other, they would first have to beat t_w which would eliminate t_w and therefore there would be no solution. So for any valid strategically optimal coalition, the problems of determining how to manipulate the competition to select the team which plays t_w in round $1 \leq k \leq n$ are disjoint. □

If no manipulations or upsets occur in a competition, there is an expected winner of each match. More formally,

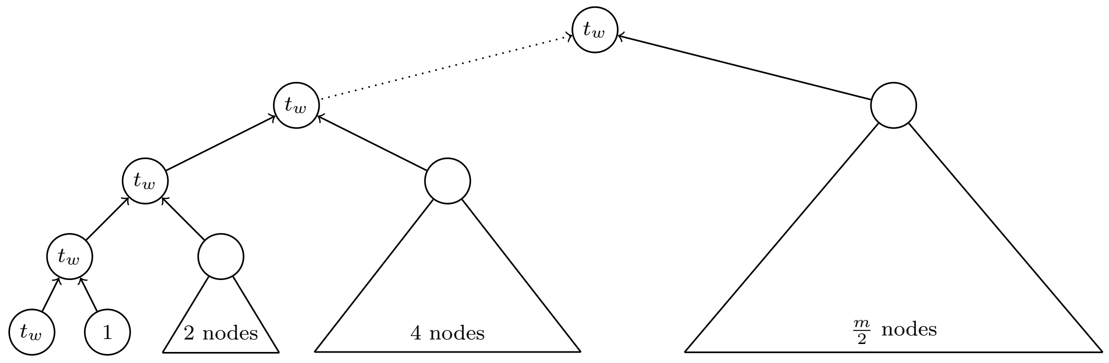


Figure 5.4: Shows the disjoint sub-problems that have to be solved in order to make t_w the winner. Each sub-problem is twice as large as the previous sub-problem.

Definition 5.3. *The expected winner of a sub-tree at round k is the team that would have reached round k if no upsets or manipulations had occurred.*

Corollary 5.1. *Given a coalition S, assuming the coalition is fixed and known (to the members) at the beginning, if t_i is the expected winner according to the tournament graph of that sub-tree at round k playing t_w and t_i is expected to lose to t_w or $t_i \in S$, then no other team in the sub-tree may be a member of S if the S is strategically optimal.*

Proof. Assume there was another coalition member $t_j \in S$ which was expected to manipulate a game in the sub-tree and S was strategically optimal. If t_j was removed from the coalition then t_i would win each of its games up to round k where they could lose to t_w either normally or by manipulating the game. Since this strategy uses at most one manipulation and any plan involving t_j would involve at least two manipulations, a contradiction is reached and S would not be strategically optimal if t_j was a member. □

If the assumption that the coalition is fixed and known (to themselves) at the beginning is relaxed, the following lemma can be used to prune teams from the strategically optimal coalition.

Lemma 5.3. *A team t that is in a sub-tree dominated by t_w—i.e., every team in the sub-tree is expected to lose against t_w—is not in any strategically optimal coalition.*

Proof. Since t_w is expected to defeat every team in the sub-tree, it is clear that any upset amongst teams in the sub-tree does not affect the results and, therefore, none of the teams could be a member of a strategically optimal coalition. □

Corollary 5.1 allows us to prune any team that belongs to a sub-tree where the root plays t_w where the expected winner is expected to lose to t_w or is a coalition member. This can greatly simplify certain problems where t_w is a relatively strong player and faces only a few challenging opponents.

The next result bounds the total number of manipulations that are possible under any tournament graph. This is an upper bound on the size of any strategically optimal coalition. First, a relaxation of the notion of strategically optimal is defined.

Definition 5.4. *A non-redundant coalition (for a team t_w) is a coalition where any manipulation strategy at round 0 has every team manipulating one game to make t_w the winner and no proper subset of the coalition could have manipulated another set of games to make t_w the winner. A team is redundant if it could be removed from the coalition and the coalition could still manipulate the tournament at round 0. A team is non-redundant if when it is removed the coalition could no longer manipulate the tournament at round 0.*

Note that all strategically optimal coalitions are non-redundant coalitions but not all non-redundant coalitions are strategically optimal coalitions. Non-redundant coalitions, and thus all strategically optimal coalitions, can only contain a bounded number of members given the size of the competition. To prove this, a few intermediate results are proved that makes the final proof of bounded size easier.

Lemma 5.4. *Given a manipulation strategy where a manipulation occurs in a specific match, an expected winner of the match t_j and a team t_i which manipulates the match in the strategy, if t_j is coalition member then t_i is redundant.*

Proof. Since t_i is the manipulator in the strategy, there must have been at least one manipulation so that t_j is no longer the winner. If t_j is a member of the coalition, t_j could have either lost normally or manipulated the game which means that under that strategy only zero or one manipulations occurs. In the strategy where t_i plays in the match and manipulates, there are at least two manipulations. Therefore, t_i is redundant if t_j is a coalition member because t_i could be removed from the coalition and the coalition could have still manipulated the competition. □

Lemma 5.5. *A coalition S contains at least one member who is redundant if two teams in the coalition manipulate two consecutive games.*

Proof. Assume there exists a situation where two coalition members manipulate two consecutive games. Figure 5.5a shows such a situation. As shown in Figure 5.5b, the earliest manipulator, t_2, could have not manipulated the first game and, instead, manipulated the winner, t_1, if t_2 defeats t_1 in a fair game. This would have resulted in one less manipulation. Alternatively, if t_1 defeats t_2 in a fair game then, as shown in Figure 5.5c, no manipulations

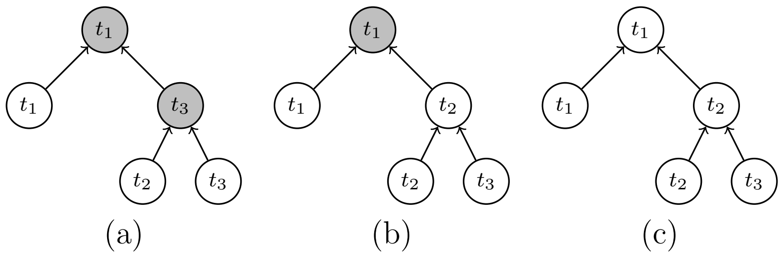


Figure 5.5: (a) A situation where t_2 manipulates the game against t_3 and then t_3 immediately manipulates the game against t_1. (b) An alternative situation where if t_2 defeats t_1 according to the tournament, then one less manipulation is need and t_3 is not a coalition member. (c) Another alternative where if t_1 defeats t_2 according to the tournament then no manipulations are needed and both coalition members are redundant.

are necessary. In either case, fewer coalition members are needed and therefore at least one coalition member is redundant. □

Using Lemma 5.5, it is possible to prove another property about non-redundant coalitions.

Lemma 5.6. *For any pair of matches where the winners play in the next round, any scenario where a manipulation occurs in both matches contains at least as many or more manipulations than a scenario where only one manipulation occurs at that round.*

Proof. Assume there are two pairs of teams where the winner of each match is facing the other team in the next round where both matches are manipulated (see Figure 5.6a, for example). Given Lemma 5.5, no manipulation can occur in the next round since both previous matches are manipulated. Therefore, one team was manipulated by a coalition into a position to lose immediately. If that coalition member had not manipulated the first game, they could have either manipulated in the second round, shown in Figure 5.6b or lost naturally, shown in Figure 5.6c. Since the number of manipulations is equal or less than the previous case while obtaining the same result, there is a scenario where equal or fewer manipulations occur if only one manipulation happens in the initial round. □

Using Lemmas 5.5 and 5.6, the following proof of the maximum size of the coalition can be obtained.

Theorem 5.1. *Given m teams, for any seeding and any tournament graph, if there exists a non-redundant coalition then it contains at most $\frac{m}{2}$ members.*

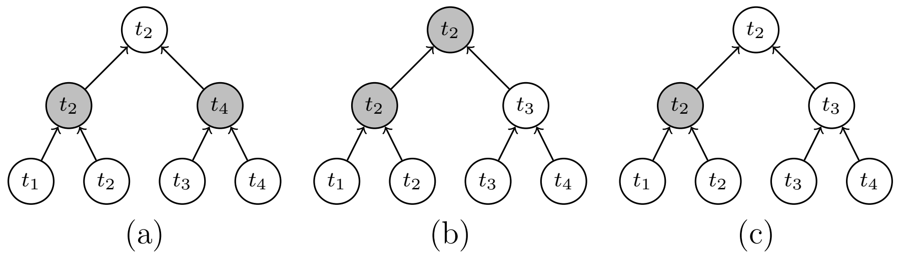


Figure 5.6: (a) A section of a cup where two manipulations happen in the same round and the victors of the matches will face each other in the next round. (b) An alternative arrangement of the manipulations which requires the same number of manipulations if t_3 defeats t_2 according to the tournament graph. (c) Another alternative manipulation strategy which requires less manipulations if t_2 defeats t_3 according to the tournament.

Proof. Proof by induction on the height of the tree n where $n = \log_2 m$. **(Base Case)** If there is a tree of height 1, there would be two teams and it takes at most one manipulation to make any team a winner. Either the team wins the game fairly or the other team manipulates so that other can win.

(Induction Step) Assume the conclusion for all i, $1 < i \leq k$. Therefore, it is necessary to prove that it holds for trees of height $k + 1$. At the level of $k + 1$, either the expected winner wins or the expected winner manipulates to make the other team win. If the team does not manipulate then there are at most 2^{k-1} coalition members in the sub-trees by the inductive assumption and, therefore, at most 2^k in the entire level $k + 1$. Therefore it suffices to show that if the expected winner manipulates, there were at most $2^{k-1} - 1$ manipulations in the sub-tree where the manipulator originated.

Observe that there are exactly $2^k - 1$ matches in a sub-tree of size 2^k. This means there are $2^{k-1} - 1$ pairs of matches and a singleton match at the top of the tree. Using Lemma 5.6, it is known that there are at most $2^{k-1} - 1$ manipulations possible in those pairs since any situation where there are two manipulations in the pair can be converted into an equivalent or better situation where there are only one manipulation among the pairs. The remaining singleton match at the top of the tree could possibly contain a manipulation but since in the next consecutive match the winner of the sub-tree manipulates, Lemma 5.5 states that this is not possible. Therefore, there are at most $2^{k-1} - 1$ manipulations possible in the sub-tree. Therefore, the induction step holds and the theorem holds for all n. □

This bound is true regardless of the tournament graph given in the problem but it is likely that the maximum size coalition given a particular seeding and a tournament graph is smaller than the given $\frac{m}{2}$. However, the following corollary which can be used to prune teams can be derived.

Corollary 5.2. *In any sub-tree of a competition at round k, a non-redundant coalition can only have at most $2^{k-1}+1$ members of the coalition and the corresponding manipulation strategy can only contain at most $2^k - 1$ manipulations for that sub-tree.*

Proof. Using Theorem 5.1, it follows that for a non-redundant coalition there can be at most 2^{k-1} coalition members to manipulate the winner of a sub-tree of height k. As well, the manipulation strategy must use at most 2^{k-1} manipulations. The winner of that sub-tree may also be a coalition member and, therefore, there can be at most $2^{k-1}+1$ coalition members in any sub-tree at round k. □

Proving a lower bound for the problem is substantially easier. The coalition must include at least the number of the expected winners to whom t_w would have lost. More formally, the following theorem states a lower bound on the number of coalition members in the coalition.

Theorem 5.2. *The coalition must include at least as many teams as the number of expected winners of each sub-problem to which t_w would lose in a fair match, where sub-problems are decomposed as in Lemma 5.2.*

Proof. If the expected winners are all coalition members then the theorem holds and if one or more of them do not belong to the coalition then it would require at least one manipulation, and therefore coalition member, to change the result. Therefore, the coalition must include as many teams as number of expected winners that t_w would lose to in a fair match. □

The minimal number of coalition members in a non-redundant coalition is at most $\log_2(m) = n$ teams where m is the total number of teams in the tournament and n is the height of the cup competition. Since strategically optimal coalitions must be non-redundant coalitions, both the upper and lower bound applies to strategically optimal coalitions.

The lower bound on the number of members of a coalition is the same whether the coalition is non-redundant or not but the upper bound on a coalition which contains redundant members is much higher. There exists a tournament graph where $m-1$ manipulations could occur and t_w wins. This can be seen easily by assuming that t_w loses to every other team. If this is so then each game can be manipulated without concern to whom t_w plays as any arrangement would require matches against t_w to be manipulated and this results in $m-1$ manipulations or, in other words, a manipulation in every game.

Given a tournament graph, it is also possible to construct the maximum number of manipulations possible to change t_w into a winner. Using the CSL algorithm from Chapter

4, construct the set of possible manipulations while keeping track of the most number of manipulations possible which is analogous to keeping the minimal number of manipulations possible. Once the possible manipulations have been constructed, compute the maximum number in the same manner.

5.2.3 An Algorithm For Determining Strategically Optimal Coalitions

The first algorithm that must be constructed is the algorithm that verifies that a coalition is a strategically optimal coalition. I show that a coalition can be verified in polynomial time (see Algorithm 5.1). Given a coalition S, the algorithm determines whether the manipulations by those teams are strategically optimal. There are $n = log(m)$ rounds in a tournament graph. At each round k, $1 \leq k \leq n$, the set of upsets U_S^k attributed to S must be part of an optimal manipulation strategy. The minimal number of manipulations needed to ensure that t_w is a winner for a given set S is denoted a_k. Fixing U_S^k as played, the minimal number of manipulations possible for rounds $k+1, \ldots, n$, denoted c_k, can similarly be determined. If, for every round k, $a_k = \left|U_S^k\right| + c_k$ then S is a strategically optimal coalition since the coalition never uses more than a minimal number of manipulations given the known upsets. Note that in Algorithm 5.1 if a coalition can no longer generate any manipulation strategy to make t_r the winner, a_k and c_k are undefined. To avoid complicating the algorithm, these cases are not shown but it should be assumed that the algorithm returns false.

Lemma 5.7. *Determining if a coalition S is strategically optimal can be computed in polynomial time.*

Proof. Calculating the minimal number of manipulations requires $O(m^2)$ time as shown in Theorem 4.3 and, for all rounds, the total time needed is $O(m^2 \log(m))$. Therefore, determining if a coalition S is a strategically optimal coalition can be computed in polynomial time. □

Algorithm FindAll can be used to determine all strategically optimal coalitions that ensure the a given team wins a tournament. It returns Null if there are no such coalitions that can ensure the team wins; otherwise it returns the set of strategically optimal coalitions (possibly the empty coalition). The initial call to the algorithm is FindAll(C, t_w), where t_w is the final winner of the tournament and C is the cup competition. The algorithm generates all strategically optimal coalitions for sub-trees and then merges them together, pruning as they are constructed. The algorithm does this by generating optimal manipulation strategies.

```
Algorithm: StrategicCoalition($C$, $t_r$, $S$, $U$)
input  : A competition tree $C$, a team $t_r$ to establish as the winner of the sub-tree,
         and a coalition $S$. Assumes that the set of upsets $U$ that occurred in the
         competition is available.
output: Returns true if $S$ is a strategically optimal coalition; false otherwise.
$Upsets \leftarrow \{\}$;
Let $n$ be the number of rounds in $C$;
for $k \leftarrow 1, \ldots, n-1$ do
    $a_k \leftarrow$ MinManipulations($C$, $k$, $t_r$, $S$, $Upsets$);
    if $k = 1 \wedge a_k \neq |S|$ then return false;
    $c_k \leftarrow$ MinManipulations($C$, $k$, $t_r$, $S$, $Upsets \cup U_S^k$);
    if $a_k < \left|U_S^k\right| + c_k$ then
        return false;
    $Upsets \leftarrow Upsets \cup U^k$;
return true;
```

Algorithm 5.1: The algorithm for verifying that a coalition S can make t_r a winner given a tournament graph G and a set of upsets U.

Pruning is then based on a coalition not establishing the team t_w (it does not achieve the goal) or it uses too many manipulations to establish the team t_w (it is not optimal). The algorithm is specified in a recursive, top-down manner.

The actual implementation uses memoization so that the result is as efficient as a dynamic programming approach but is somewhat easier to read and understand. The memoization is not shown in the algorithm specification, but the idea is to, prior to each recursive call, check whether the result has been previously computed. If it has, the stored result is used. If it has not, the recursive call is executed and then the result is memoized (see [19, p.347-349] for further details on dynamic programming using memoization).

Two optimizations are not shown in the algorithm: (i) if t_r, which is expected to win the sub-tree C_r, is expected to win against the winner of the subtree C_o, the **foreach** loop is avoid as the empty coalition will be the result; and (ii) at the end of each iteration of the **foreach** loop, if S_o contains the empty coalition, every other coalition will be non-minimal so the algorithm breaks out of the loop.

Example 5.3 (Finding A Strategically Optimal Coalition)**.** *Referring to the cup competition described in Example 5.1 and the manipulations illustrated in Figure 5.3, there is one strategically optimal coalition $\{t_{13}, t_{16}\}$. Decomposing the tree as in Lemma 5.2 and applying Corollary 5.1, it can be seen that the expected winner of the first three sub-trees—t_1, t_3 and t_5—are the actual winners and therefore there are no coalition members in those sub-trees. All that remains to be shown is how the strategically optimal coalition is generated*

```
Algorithm:FindAll($C$, $t_r$)
input  : A competition tree $C$ and a team $t_r$ to establish as the winner. Assumes
         that $t_w$, the final winner of the tournament, is available.
output: Returns all strategically optimal coalitions that can ensure $t_r$ wins the
         subtree $C$. Returns Null if there are no such coalitions; otherwise, returns
         the set of coalitions (possibly the empty coalition).
if $C$ consists of a single team then
    if $T$ contains $t_r$ then return $\{\}$;
    else return Null;
else
    Let $C_r$ be the subtree of $C$ that contains team $t_r$ and let $C_o$ be the other subtree;
    $S_r \leftarrow$ FindAll($C_r$, $t_r$);
    if $S_r$ = Null then return Null;
    $S_o \leftarrow$ Null;
    foreach $t_k \in C_o$ do
        $S_{temp} \leftarrow$ FindAll($C_o$, $t_k$);
        if $S_{temp}$ = Null then return Null;
        if $t_r$ is not expected to win against $t_k$ then
            add $t_k$ to each coalition in the set of coalitions $S_{temp}$;
        add the coalitions in $S_{temp}$ to $S_o$;
        remove from $S_o$ any coalition that is a superset of another coalition in $S_o$;
    $S_{temp} \leftarrow S_r \times S_o$;
    if $t_r = t_w$ then // $t_w$ is to win the subtree $C$
        foreach $S \in S_{temp}$ do
            if $\neg$StrategicCoalition($C$, $t_r$, $S$) then
                prune $S$ from $S_{temp}$;
    return $S_{temp}$;
```

Algorithm 5.2: Generates for a given cup competition all of the possible strategically optimal coalitions which establish t_w. The recursive call generates all of the possible strategically optimal coalitions in each subtree and then the results are combined.

from the right most sub-tree. This is done by building and verifying each coalition is not a strategically optimal coalition while pruning as many possibilities at each stage.

5.2.4 Experimental Results

Given that no real world data exists detailing large scale coalitional cheating, it is necessary to construct synthetic data to test the algorithm. As in Chapter 4, the statistical data related to upsets in the NCAA Division One Basketball Championship (see Figure 5.7) is used to randomly model real world upsets in a cup competition. The tournament graph is constructed by examining each pair of games and generating the winner from the random probability distribution. The resulting outcomes of the tournament constructed from the tournament graph should approximate the real world outcome of the tournament. Upsets are generated in a similar manner as the tournament. The teams are seeded using pools of sixteen teams and the best-plays-worst paradigm. Each match is then played according to the tournament with a random chance sampled from the distribution of an upset occurring. To test the algorithm, it is necessary to embed a coalition which is actively changing the tournament so that a specific team wins the tournament and reacts to the upsets as the upsets occur in the tournament. These coalitions are generated using a heuristic method described in Algorithm 5.3. Once a coalition is generated, additional upsets are added to the competition according to the upset model shown in Figure 5.7. This method embeds the optimal manipulations of coalitions and adds extra manipulations randomly as described by the model. The only change is that care is taken to ensure that the new upsets do not cause the coalition to become nonviable as the method is to detect successful coalitions.

```
Algorithm:GenCoalition($T$,$G$,$t_w$)
input  : A set of teams $T$, tournament graph $G$ and a team $t_w$.
output: Returns a coalition of teams which ensures that $t_w$ wins.

current ← $T$;
changed ← True;
newCoalition ← minimal subset of current needed to make $t_w$ a winner in $T$;
while changed do
    changed ← False;
    foreach team ∈ newCoalition ∧ team ∉ checked do
        remove team from current;
        if $t_w$ can win $T$ given current is the coalition then
            nextCoalition ← minimal subset of current needed to make $t_w$ a winner in
            $T$;
            changed ← True;
        else
            add team to current;
            add team to checked;
    newCoalition ← nextCoalition;
return newCoalition;
```

Algorithm 5.3: The algorithm takes a set of teams T, a tournament graph G and a desired winner t_w and generates a coalition that manipulates the coalition optimally to make t_w the winner.

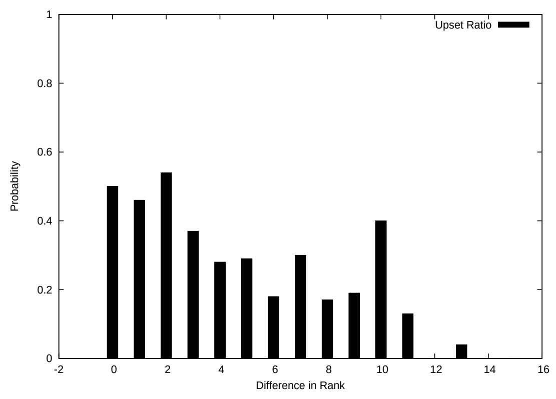


Figure 5.7: The probability, constructed from the previous twenty five NCAA Division One Basketball Championships (1985-2009), that a team whose rank is i positions lower than another team would upset that team.

To compare strategically optimal manipulations from random upsets, 1000 instances with generated coalitions and 1000 instances with random upsets were tested. The results, shown in Figure 5.8, show that random upsets rarely produce strategically optimal behaviour and it was hypothesized that the existence of strategically optimal behaviour meant that there was a manipulating coalition. To test the hypothesis, tests were rerun on 1000 new instances with generated coalitions and 1000 new instances with random upsets. Table 5.2 shows the accuracy, precision, recall and F-measure of the new instances. These results show that overall the accuracy of the method is high but that the results are weaker on smaller sized tournaments where accuracy is 76.7%. From the recall values, it can be observed that no false negatives are produced and all of the instances with coalitions are identified as having coalitions. There are two problems. The first is the false positives where the algorithm concluded a coalition exists where none exists and the second is that even when a coalition is identified as existing there can be quite a large number of candidates that could be the potential cheating coalition. These problems are discussed below.

From the precision values, it is clear that the problem of false positives is worse when the tournaments are small but it can be seen in Table 5.3 that the size of coalitions and the number of different coalitions is small when tournaments are small. Therefore, it is practical to evaluate the small number of small coalitions to determine, by reviewing the game tapes, if a manipulation occurred. However, as tournaments grow, this problem becomes more pronounced and the number of potential strategic coalitions increases. Table 5.4 shows

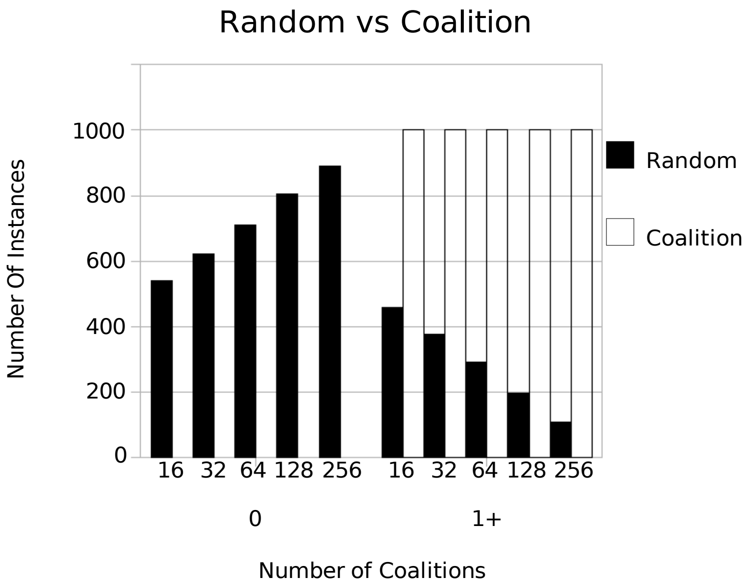


Figure 5.8: For various tournament sizes, the number of instances where a strategically optimal coalition is detected where the upsets in the instances are generate at random or by a strategically optimal coalition. 1000 random instances and 1000 instances with strategically optimal coalitions are tested.

the precision, recall, and F-measure values when comparing the set of strategically optimal coalitions returned when there exists a coalition. Referring to the precision values as the size of the tournament grows, there are more strategically optimal coalitions in the larger tournaments. However, results in Table 5.5 show that the percentage of teams from the entire pool of teams drops as the size of the tournament increases and, in most identified coalitions, nearly 60% of the teams are actually members of the coalitions. Furthermore, in most instances, the percentage of falsely accused teams amounts to less than 6% of the total teams in a competition. These results show that the large number of strategically optimal coalitions are due to a multiplicative effect due to a small number of misidentified cheaters.

It is assumed that it is practical for a person to evaluate the individual game results to determine if cheating exists when suspected. Note that human evaluation allows certain matches initially viewed as upsets to be reclassified as manipulation or as a true upset. This reclassification could remove a team from all strategically optimal coalitions or require the team be a member of the strategically optimal coalition. In this way, this method could be

Table 5.2: The accuracy, precision, recall and F-measure values when comparing 1000 instances with coalitions and 1000 random instances with no manipulation.

Size	Accuracy	Precision	Recall	F-measure
16	0.767	0.682	1.000	0.811
32	0.812	0.726	1.000	0.842
64	0.854	0.773	1.000	0.873
128	0.894	0.825	1.000	0.904
256	0.935	0.884	1.000	0.939

paired with other current methods which use betting to identify individual games where teams are cheating.

Given a set of strategically optimal coalitions, it is likely that the human observer would prefer to start with a good candidate for the actual strategically optimal coalition. One possible criteria is to select the coalition that has the most number of teams actually manipulating a game. The reason why this may be a good criteria is the implicit assumption that it would be less likely for a large number of random upsets to happen optimally. Table 5.6 shows the accuracy of the method when selecting the best coalition using the most games manipulated by the coalition criteria. The results show, rather unsurprisingly, that detecting coalitions which actually manipulate a large number of games have high accuracy and that detecting coalitions smaller examples have higher accuracy than detecting coalitions from larger examples. The results do show that, with the exception of instances of size 256, even as the number of manipulating coalition members decreases the accuracy remains relatively high even though the precision results from Table 5.4 show that there is a great number of false positive coalitions to compare against. These results suggest that starting with the coalition that contains the most number of upsets would be a worthwhile strategy.

5.3 Detecting Manipulations in Round Robins

In round robins, there are two types of manipulations that a coalition can perform to change the result of the tournament. The first is the most simple and involves a coalition member losing its games directly to the desired winner, t_w. The second complex manipulation is to lose games amongst themselves so that t_w does not have to win more games in order to win the competition. The problem with the second type of manipulation is their motivation. Though it may not be immediately obvious, if a coalition of teams needs to use the second type of manipulations to successfully execute a strategy then the expected winner of the

Table 5.3: The size and number of the coalitions generated in the smallest 10, 25, 50, 75 and 95 percent of instances. The results show the comparison between 1000 random and 1000 generated instances.

	Generated						Random					
	Size of Coalitions						Size of Coalitions					
	10%	25%	50%	75%	95%	100%	10%	25%	50%	75%	95%	100%
16	1	2	3	4	5	7	0	0	0	1	2	4
32	2	3	4	6	8	11	0	0	0	1	3	6
64	3	5	7	9	12	16	0	0	0	2	4	7
128	6	8	11	13	16	23	0	0	0	4	6	11
256	10	13	16	20	24	33	0	2	5	7	11	16
	Number of Coalitions						Number of Coalitions					
	10%	25%	50%	75%	95%	100%	10%	25%	50%	75%	95%	100%
16	1	1	1	1	2	6	0	0	0	1	2	4
32	1	1	1	2	4	18	0	0	0	1	2	5
64	1	1	2	4	8	36	0	0	0	1	3	18
128	1	2	4	10	42	510	0	0	0	0	5	60
256	2	6	20	72	340	4515	0	0	0	0	12	432

tournament is one of the coalition members. The reason for this is the assumption that a team can only throw games and thus not win them when they should lose. Given this restriction and assuming that the expected winner was not part of the coalition, reducing the points among themselves when the expected winner is not losing any points serves no practical purpose as t_w still would need to earn more points than the expected winner.

Using the definitions from Section 2.5, it is again assumed that the results of the competition are known and the coalition detection occurs with a known set of upsets and results. It is also assumed that if two coalition members are playing each other or one coalition member is playing t_w then any upset is intentional and a manipulation. However, an upset of a coalition member by a non-coalition member is considered to be just an upset. In Section 5.2, it was assumed that this would not occur but additional upsets in round robins often do not have huge negative consequences such as the elimination as in the case of cup competitions. Another assumption made in this work is that a coalition member will always manipulate any game they play against t_w so that t_w wins the competition unless t_w has clinched without further action by the coalition. Discussion of round robins in this chapter are limited to single round robin tournaments using the simple win-loss model.

Example 5.4. *To illustrate the concept of detection in round robins, a complete examination of the optimal manipulation strategies of round robins of size four is made. Round robins of size four present an interesting case because no complex manipulations are possi-*

Table 5.4: Shows the precision, recall and F-measure values for the Strategically Optimal Coalition results on problems of size 16, 32, 64, 128 and 256 over 1000 instances.

Size	Precision	Recall	F-measure
16	0.833	1.000	0.909
32	0.553	1.000	0.713
64	0.312	1.000	0.475
128	0.090	1.000	0.166
256	0.012	1.000	0.023

ble. There are four possible outcomes of a round robin of size four, which are $\{3,2,1,0\}$, $\{3,1,1,1\}$, $\{2,2,2,0\}$ *and* $\{2,2,1,1\}$, *where the teams are sorted by the number of wins earned in descending order. In the first outcome, the team that earns three wins must lose a game for any other team to win. This is optimal if the teams with either two or one win are the desired winner. If the team with zero points is the desired winner, then the team that earns three points must form a coalition with exactly one of the teams that earns two or one points. For the second outcome, the only valid optimal strategy is for the team that earns three wins to lose to the desired winner. For the third outcome, only the team that wins no games is not already in a winning position and exactly two of the other teams that earn two points must lose to the team that earns no points to optimally change the outcome. For the fourth outcome, one of the teams that earn two points must lose to one of the teams that earns one point. All other manipulations are non-optimal. Thus, any detection method should highlight any team or set of teams which executes the optimal manipulations.*

5.3.1 Coalitions Formed by Only Losing to Desired Winner

Since the first type of manipulation where the only manipulations that occur are to t_w is simpler and more likely, it is interesting to look at the restricted form of the problem where only this type of manipulation occurs. The goal of any coalition S which only manipulates games against t_w is to lose enough games to t_w so that t_w earns more victories than the expected winner. In this context, a strategically optimal coalition would be a coalition that always manipulates games to t_w while t_w is not expected to win and does nothing otherwise. Given this goal and the tournament graph $G = (T, E)$ where T is the set of teams, the number of victories can be worked out as the difference between the expected number of victories of the expected winner, ew_e, and the expected victories of t_w, ew_w. If every team tied as the expected winner is a member of the coalition and can lose a game to t_w, then one less point is needed to make t_w a winner. It is assumed that if t_w is tied

Table 5.5: The average percentage of teams that are identified as being members of a strategically optimal coalition, the average percentage of the teams identified correctly as members of a strategically optimal coalitions and the average percentage of teams misclassified as members of a strategically optimal coalition. Each result shows the percentage value for the top 10, 25, 50, 75 and 95 percent of results taken from 1000 instances.

	Percentage of Teams Identified as Members of Strategically Optimal Coalitions						Percentage of Teams Identified Correctly as Members of the Coalition					
	10%	25%	50%	75%	95%	100%	10%	25%	50%	75%	95%	100%
16	6.2	12.5	18.8	25.0	37.5	50.0	100.0	100.0	100.0	100.0	75.0	33.3
32	6.2	9.4	15.6	21.9	28.1	40.6	100.0	100.0	100.0	87.5	66.7	33.3
64	4.7	9.4	14.1	18.8	23.4	31.2	100.0	100.0	91.7	83.3	68.2	33.3
128	3.9	7.0	10.9	14.8	19.5	25.0	100.0	93.1	87.1	78.6	63.8	33.3
256	3.9	6.2	9.4	12.5	16.4	21.5	94.1	89.1	82.0	73.6	59.8	28.6
	Percentage of Teams Misidentified as Members of the Coalition											
	10%	25%	50%	75%	95%	100%						
16	0.0	0.0	0.0	0.0	4.7	13.9						
32	0.0	0.0	0.0	2.2	6.0	13.2						
64	0.0	0.0	1.0	2.3	4.6	9.8						
128	0.0	0.5	1.3	2.5	4.8	9.0						
256	0.3	0.8	1.7	2.8	4.8	8.6						

with the other leaders then t_w would win given some set of tie-breaking criteria.

$$w_w = \begin{cases} ew_e - ew_w & \text{if } \exists t_i \ (ew_i = ew_e \text{ and } t_i \notin S) \\ ew_e - ew_w - 1 & \text{if } \forall t_i \ (ew_i = ew_e \text{ and } (t_i, t_w) \in E) \text{ and } t_i \in S) \end{cases} \tag{5.1}$$

There are two parts to determining the set of strategically optimal coalitions which only lose games to t_w. The first step is to identify all of the potential coalition members and the second step is to generate all the feasible coalitions from those potential members.

Algorithm 5.4 shows the steps for generating all of the possible members. The first step is to remove all teams that would lose to t_w naturally since these teams could not add any positive benefit to a coalition that loses games to t_w as they are expected to do that. The second step is to remove all of the teams that win games that they were not expected to

Table 5.6: Compares the best coalition returned by the coalition detection method against the actual coalition. The best coalition is defined to be the coalition with the most number of coalition members which actually manipulate. Shows the accuracy of the method for each size of problem on 1000 instances. Each column represents the accuracy of the method when tested on coalitions where at least X% of the teams manipulates.

	100	90	80	70	60	50	40	30	20	10	0
16	100.0	98.4	98.4	97.7	96.9	96.5	95.2	95.1	95.0	94.9	94.9
32	100.0	98.8	97.5	94.9	92.4	90.0	86.9	85.0	84.3	84.0	84.0
64	100.0	97.7	94.7	89.5	81.6	77.9	72.8	70.0	68.1	67.1	67.1
128	100.0	96.3	91.0	82.6	74.3	65.2	58.1	54.1	52.3	51.7	51.4
256	100.0	95.8	90.0	75.5	58.3	45.6	39.3	33.9	30.8	30.0	29.8

win. Since a coalition member would prefer for games to stay the same and not generate extra upsets, the coalition member should always lose every game they are expected to lose. This leaves the teams that did not win any extra games and were expected to defeat t_w. These teams are broken into two sets: those teams that actually lost to t_w and the those teams that actually defeated t_w. The first set is denoted A and the second set is denoted B. Teams in both of these sets play t_w at different points during the round robin. Specifically, depending on whether t_w is expected to win, a coalition member should lose a game to t_w or not. Therefore, a team in A is optimal if and only if t_w is not expected to win the coalition and further manipulation is required and a team in B is optimal if and only if t_w is expected to win and no further manipulation is expected to be required. Pruning the teams that are not optimal from the sets A and B, any combination of the remaining teams where the number of teams is equal to w_w would be a valid strategically optimal coalition.

Lemma 5.8. *Algorithm 5.4 computes the set of possible coalition members in $O(m^3)$ time in the worst case.*

Proof. The first step of removing all teams that lose naturally to t_w can be done in time $O(m)$, where m is the number of teams. It is valid to remove those teams since they could not actually manipulate a game. This is also true for finding the set of teams that earn extra points by upsetting teams. Removing teams that win extra games is valid because those teams generate extra upsets and thus the strategy would not be minimal if they were included. The remaining set of teams can be split into the sets A and B in linear time. Determining if t_w is the expected winner for m rounds given $O(m^2)$ games takes $O(m^3)$ time. Each set can be pruned of the remaining non-optimal members in linear time. The remaining teams are non-optimal because if t_w was expected to win then no manipulation

```
Algorithm: PossibleCoalitionMembers($R$, $G$, $t_w$)
input : A round robin competition $R$, a tournament graph $G = (T, E)$ where $T$ is
        the set of teams and a team $t_w$ which wins the competition.
output: Returns all possible coalition members which behave optimally along with
        constraints on a specific subset of the results.
// removes all teams that are expected lose to $t_w$ or win extra games
$A \leftarrow$ the set of all teams that are upset by $t_w$ and win no extra games;
$B \leftarrow$ the set of all teams that defeat $t_w$ and win no extra games;
// Identify dates when $t_w$ is expected to win
foreach round $r$ of the season do
    Identify whether $t_w$ is the expected winner in round $r$;
// remove all teams that do not win or lose where appropriate
Remove all teams from $A$ that played $t_w$ in round $r$ where $t_w$ was expected to win in
round $r$;
Remove all teams from $B$ that played $t_w$ in round $r$ where $t_w$ was not expected to
win in round $r$;
return $A$ and $B$;
```
Algorithm 5.4: Generates, for a given round robin competition and tournament, all of the possible coalition members which act optimally to make t_w a winner.

would have been better than a manipulation and if t_w is expected to lose then further manipulation is necessary and it is always better to manipulate as early as possible to deal with potential upsets later. Therefore, the worst case complexity of the algorithm is $O(m^3)$. □

Given the set of teams A and B, the teams must be combined into coalitions that could optimally manipulate the tournament. For a coalition to be strategically optimal, all of the coalitions must be of size w_w, calculated as in Equation 5.1, and there should always be enough possible coalition members to ensure that t_w will become the winner. Any set of teams of size w_w from the set $A \cup B$ that satisfies the constraints is a possible strategically optimal coalition which only loses games to t_w. There are potentially an exponential number of sets which could satisfy the conditions to be strategically optimal coalitions.

5.3.2 Coalitions Formed by Losing to Any Team

It is possible for coalitions to form where top teams lose to weaker teams in order to have another team win. Possible motives include monetary payoffs, improved draft standing and prior obligations such as, for example, a 'quid pro quo' situation where top teams take turns winning the championship every year to ensure they maintain the lion's share of the profits. Unlike the previous manipulation strategies where only losses to t_w could occur, coalitions that must self sacrifice themselves would manipulate other games.

Some of the constraints from the previous section must be relaxed. Specifically, the constraint that teams which earn extra points are not part of the coalition must be relaxed and the constraint that teams that lose naturally to t_w are not part of the coalition must be relaxed. The first constraint must be relaxed so that teams which earn extra victories when t_w is an expected winner are not part of the coalition. The second constraint is removed entirely. However, an additional constraint is added: all teams that earn extra victories and are part of the coalition must have earned the extra points from coalition members. Given a Boolean variable s_i which is true if t_i is a member of the coalition and the set of teams, L_i, that lose to t_i when they should have won, the following constraint can be derived,

$$\forall t_i \in T \quad s_i \Rightarrow \bigwedge_{s_j \in L_i} s_j \ . \tag{5.2}$$

The set of teams A and B can be calculated using Algorithm 5.4 except no check must be made to ensure they earn no extra games. An additional set C is defined that includes teams that were expected to lose to t_w and did lose to t_w while only receiving wins when t_w is not the expected winner. Given that a coalition could effectively achieve their goal by only manipulating games against t_w, there must be at most $w_w - 1$ members from $A \bigcup B$ for complex manipulations to be necessary. Let u be the number of teams from $A \bigcup B$ belonging to the coalition such that $0 \leq u < w_w$. Given a value of victories that t_w could achieve, u, there is a specific set of top teams, TT_u, that must lose at most $w_w - u$ games on purpose. Since any team can be upset, a minimal manipulation strategy is one where the coalition only manipulates a minimal number of games between themselves and do not earn any wins from teams not in the coalition. Since coalition members are attempting to lose games to allow t_w to exceed in wins them and those teams do not earn extra victories unless necessary, any upset only reduces the number of manipulations that must be made. An upset may make a coalition unviable by causing a non-coalition member to have more points than t_w could be expected to make even under the coalition.

Since each TT_u set can be solved in isolation, the goal is to extend the set if necessary so that the coalition can achieve the goal but never loses more games than necessary amongst themselves. After identifying the set of teams that form the sets A, B, and C, all valid sets must be generated. The following CSP is used to find all possible coalitions from A, B and C given the external constraints. Constraint 5.3 ensure the coalition only attempts to have t_w earn enough games for TT_u to be valid. Constraint 5.4 is the extra constraint described above that enforces that teams that are upset by coalition members are coalition members themselves. Constraint 5.5 states that teams that are not a member of the possible sets of teams are not part of any coalition. Constraint 5.6 ensures that all the teams in the TT_u set under consideration are members of the coalition.

$$\sum_{t_i \in A \cup B} = u \ , \tag{5.3}$$

$$\forall_{t_i \in T} \quad s_i \Rightarrow \bigwedge_{s_j \in L_i} s_j \ , \tag{5.4}$$

$$\forall_{t_i \notin A \cup B \cup C} \quad \neg s_i \ , \tag{5.5}$$

$$\forall_{t_i \in TT_u} \quad s_i \ . \tag{5.6}$$

Given the set of possible coalitions generated by solving the CSP, it must be ensured that only a minimum number of manipulations are used in each round to ensure the coalition is strategically optimal. Using the minimum cost feasible flow technique described in Section 4.3 for finding minimal manipulations in round robins, it is determined if a smaller set of teams could have manipulated the result at the start of the competition. If so, the possible coalition is discarded. Next, the remaining coalitions are checked to ensure that the manipulations used in any given round are minimal by calculating the before and after cost of using the actual manipulations, which should be the same for every round. Any possible coalition that has a minimal number of members and uses as few manipulations as possible in each round is strategically optimal. A minimal number of manipulations in this case refers to manipulations between coalition members as coalition members could be upset by non-coalition members without manipulating.

5.3.3 Experimental Results

To test detectability of the two different types of round robin manipulations, it is necessary to create a set of test problems. First, a set of round robin schedules must be generated such that each team plays every other team. A canonical schedule was generated as described by de Werra [20] and then the teams were permuted to generate different schedules. With each round robin, a corresponding tournament graph was generated using data from the NCAA Division I Basketball Championship (see Figure 5.7). Problems were generated for round robins of even size from 4 to 40. Forty was chosen as the maximum as it was larger than the number of teams in most professional sports leagues. For clarity, abbreviated tables with results for round robins of sizes six, twelve, 18, 24, 30 and 36 are presented in this chapter and the complete tables can be found in Appendix C.

One hundred random instances were generated along with one hundred instances with coalitions embedded that only use manipulations against t_w. Table 5.7 shows the accuracy of the method for detecting manipulations. The accuracy of detecting simple manipulations only against t_w is better than 88% regardless of size and better than 97% on instances with ten or more teams. When a team is correctly identified, the set of teams identified as being members of the coalition are almost always members of the coalition. Table 5.8 that there are at most three extra teams in $A \cup B$ and on average there is less than one extra team in $A \cup B$. Given the experimental assumptions about the tournament graph and upset probabilities, this means that the method can determine a difference between random and strategically optimal coalitions and the difference between the possible set of members and the actual set of members is small.

Table 5.7: The effect of round robin size on the accuracy of identifying manipulation of round robin competitions by only manipulating games against t_w. Table C.1 in Appendix C gives the complete results for all even sized round robins from 6 to 40.

6	12	18	24	30	36
88.5	97.5	99.5	99.5	99.0	99.5

Table 5.8: The effect of round robin size on the minimum, maximum and average difference between the expected coalition and $A \cup B$ where the coalition is a subset of $A \cup B$. Table C.3 in Appendix C gives the complete results for all even sized round robins from 6 to 40.

6		12		18		24		30		36	
range	average	range	average	range	average	range	average	range	average	range	average
[0–2]	0.34	[0–1]	0.18	[0–1]	0.16	[0–2]	0.19	[0–3]	0.10	[0–2]	0.09

The same experiment was performed for complex manipulations where the top teams lose games to ensure that t_w wins. The one change is that for round sizes six, eight, ten and twelve there were fewer than one hundred coalitions with complex manipulations embedded. The reason for this was that it was not possible to have complex manipulations in some of the smaller round robins given the tournament graph. In those four cases, there are actually 69, 91, 99 and 99 examples where there existed a strategically optimal coalition embedded in the upsets, respectively. The accuracy of detecting complex manipulations from random upsets is better than 97.5% over all sizes. Table 5.9 shows the accuracy of detection for a selection of round robin sizes. For instances with embedded upsets, the coalitions that are generated from the CSP tend to be very similar to the actual coalition. In many cases, only the single correct coalition is generated and, if spurious coalitions are generated, the coalitions are at most five different teams and much less than one on average.

Table 5.10 shows the minimum, maximum and average difference in coalition membership for a selection of teams. The number of spurious coalitions generated can be quite large, in one case as many as 38 extra coalitions, but on average across all sizes of round robins less than 1.72 spurious coalitions are generated. Table 5.11 shows the minimum, maximum and average number of coalitions found to be strategically optimal for a selection of teams.

Table 5.9: The effect of round robin size on the accuracy of identifying manipulation of round robin competitions where top teams must lose. Table C.2 in Appendix C gives the complete results for all even sized round robins from 6 to 40.

6	12	18	24	30	36
98.2	100.0	98.0	100.0	100.0	100.0

Table 5.10: The effect of round robin size on the minimum, maximum and average difference between the generated coalitions and the expected coalition. Table C.4 in Appendix C gives the complete results for all even sized round robins from 6 to 40.

6		12		18		24		30		36	
range	average	range	average	range	average	range	average	range	average	range	average
[0–2]	0.32	[0–2]	0.16	[0–4]	0.48	[0–4]	0.64	[0–3]	0.67	[0–3]	0.70

Table 5.11: The effect of round robin size on the minimum, maximum and average number of generated coalitions for instances where a coalition was manipulating the competition. Table C.5 in Appendix C gives the complete results for all even sized round robins from 6 to 40.

6		12		18		24		30		36	
range	average	range	average	range	average	range	average	range	average	range	average
[1–3]	1.24	[1–1]	1.00	[1–2]	1.08	[1–21]	1.47	[1–39]	1.89	[1–22]	2.72

A number of extra teams were identified as winning via a strategically optimal coalitions. When there exists a strategically optimal coalition, there can be other teams that are identified as being made winners via a strategically optimal coalition. However, no more than 37% of instances had two winners identified as being strategically optimal and, for most round robin sizes, the percentage is much smaller. Table 5.12 shows the percentage of instances where extra teams have been identified as winning via a strategically optimal coalition for a selection of round robin sizes.

Table 5.12: The effect of round robin size on the percentage of instances with two or more teams identified as having won through strategically optimal coalitions using simple manipulations directly to t_w or complicated manipulations involving the top teams. Random instances are not shown as this occurred in only two simple random instances and no complex instances. Table C.6 in Appendix C gives the complete results for all even sized round robins from 6 to 40.

	Percentage of Instances with Multiple Winners	
Size	Simple	Complex
6	37	6
12	27	12
18	25	20
24	25	17
30	24	17
36	29	11

5.4 Detecting Seeding Manipulations

In previous sections, I examined the detection of manipulations of a competition by a coalition of teams by observing the pattern of game results. In this section, I focus on detecting manipulations of the seeding of a cup competition by the scheduler. For detection, it is assumed that, as in many sports leagues, the teams have a ranking criteria which is used to make pairings and the detectors have some notion of the ranking. In Section 4.4, the problem of finding a seeding manipulation strategy by the scheduler was discussed and four different types of restrictions were introduced: pooling restriction, team arrangement restriction, criteria modification restriction and pool arrangement restriction. These restrictions were combined to create families of restrictions which could be used to classify various sporting competitions. This section focuses on showing the detectability of the different families of restrictions.

When a restriction is not enforced for a particular competition, a fair tournament is assumed to have a ranking which differs little from the known ranking and unfixed team arrangements, poolings and pool arrangements are assumed to be generated at random.

The justification for this assumption is that a fair tournament would have teams that are placed given their real ranking. Additionally, an unfixed seeding should not favour any team over any other team allowing the best teams the chance to rise to the top. Lastly, pooling should be done randomly as any pool or pool arrangement should be equivalent given the team arrangement and rank criteria restrictions.

As with the work in previous sections, there are other methods which could be used to detect seeding manipulation like observing external factors like betting patterns or game specific play. These methods are outside the scope of this work though each could be used in concert with the techniques described here.

The families of restrictions can be partitioned based on whether it is possible to detect a seeding manipulation strategy based on the resulting seeding. The following two sections describes which families of restrictions are in which partition and why.

5.4.1 Families of Restrictions Where Detection is Not Possible

The $011X$ family of restrictions requires that the pair arrangement and rank remain fixed but allows any arrangement of pools and pool winners. Since it is assumed that a fair pooling and arrangement of the pool winners happens randomly, any seeding that ensures the pair arrangement and ranking are unchanged could possibly have been generated by a random draw. Therefore, from the pattern of the seeding, there is nothing to distinguish a manipulated seeding from an fair seeding and detection of this type is not possible. Since the 1110 and 1111 families of restrictions have the pair arrangement restriction and fixed ranking, the argument applies to those families as well and detection from the pattern of the seeding is not possible.

The $10X0$ family of restrictions does not have a restricted pair arrangement and may or may not have a fixed ranking. However, detection from the pattern of the seeding is not possible in this case either. Since the pair arrangement is not restricted, the teams within a pool can be arranged in any manner and a fair arrangement of the teams would be randomly. Since any arrangement is possible under a random draw of the arrangement, there is nothing to distinguish a manipulated seeding from a random seeding. Since the $10X1$ family of restrictions has fixed pooling and unfixed team arrangement, the argument applies to this family as well and detection from the pattern of the seeding is not possible.

The $00XX$ family of restrictions has no restriction on the seeding. A fair seeding would be one drawn at random and therefore any manipulated seeding would be possible under the restriction. As such, detection from the pattern of the seeding is not possible.

5.4.2 Families of Restrictions Where Detection is Possible

In this section, the detectability of families of restrictions $010X$, 1100, and 1101 is discussed. Recall from Section 4.4, that if the ranking was completely unfixed then the problem is equivalent to the problem with both the pair arrangement and ranking criteria constraints unfixed. Therefore, the discussion is limited to problems where the ranking criteria is unfixed but bounded.

Detection of manipulation in models $010X$, 1100, and 1101 with fixed team arrangement but unfixed, but bounded, ranking criteria is aided by the fact that team arrangement is linked to a specific rank. Under the assumption that the detectors can reasonably construct the ranking of teams in the tournament, the difference between ranking of teams in seeding, known from the team arrangement, and the actual ranking provides a significant indicator of manipulation. Coleman et al. [17] discussed the bias of selection in the NCAA Division One Basketball championship where the authors concluded that the bias was due to in part the discrepancy between the perceived ranking and actual ranking. There are many different ranking measures that could be used to construct a likely ranking of teams and compared against the results [11, 4, 15]. One important thing to note here is that the quality of the detection is highly dependent on the quality of the assumed ranking and, therefore, is only as good as the ranking model used which are primarily heuristic in nature. One other measure that could be used to determine if an actual manipulation occurred would be if the expected winner changed due to the discrepancy in the ranking.

5.5 Summary

The first part of this chapter focuses on detecting strategically optimal coalitions in cup competitions. Strategically optimal coalitions are coalitions that contain no redundant members and change their strategy to handle changes due to unexpected upsets in the tournament using as few manipulations as possible. A dynamic programming approach is proposed along with a set of pruning conditions that successfully scales to competitions with 256 teams. Experimental results show that the method can distinguish the presence of manipulating coalitions with accuracy between 76.7% and 93.5% depending on the size of the competition. For small sized competitions, it is shown that both the size and number of coalitions is small and, therefore, any false positives could be checked. In competitions of larger sizes, the detection algorithm returns a great deal more possible coalitions when a coalition exists. However, analysis of the false positives shows that extra coalitions tend to have a large overlap with the actual coalition. As well, in ninety-five percent of cases at most 6% of the total teams are falsely placed in a strategically optimal coalition which allows a person to effectively weed out teams from the competition by observing the individual games for manipulation. Selecting the coalition which actually manipulates the most number of games has promise as discriminating heuristic as the selected heuristic has a high correlation with the actual coalition and the overlap is small in all but the largest of tournaments. This provides a good starting place for the people attempting to detect cheating.

The second topic discussed in this chapter discusses how to detect manipulation of round robin tournaments. Two possible types of manipulation are discussed: direct manipulation of t_w and manipulation of other teams. Given the model of upsets and the randomly generated tournament graphs, it was determined that random upsets could be distinguished from upsets where a strategically optimal coalition was generating results. If the teams were using just manipulations to t_w then the accuracy of the technique is better than 88% and this percentage improves when distinguishing between random and more complicated manipulations with an accuracy better than 97%. In both cases, spurious possible coalitions are generated by the methods but the average number of extra members is less than one. The one issue with the technique is that extra possible strategically optimal winners are found but at worst for a single small size round robin 37% of instances were identified with multiple winners.

The last topic covered in this chapter is the discussion of detecting manipulation in seeding. For many families of restrictions, the pattern of the seeding does not differ between manipulated and fair seedings. As such, detection based on the pattern of the seeding is not possible for those seedings. However, if it can be observed that the ranking has been changed then detection is possible. The issue for these methods of detections, which have been used in practice [17], is that if the known ranking is poor then the quality of the detection suffers.

In the next chapter, some final concluding remarks are made and summary of results from the entire thesis is presented.

Chapter 6

Conclusions and Future Work

This thesis looked at three different computational problems in sports. The first problem was a qualification and elimination problem applied to the playoff system of the National Hockey League and solved with a combination of network flows, enumeration and constraint programming. The second problem concerned the complexity of different manipulation problems in sports by applying techniques from election manipulation and winner determination research. Manipulation of seeding mechanisms was evaluated experimentally using constraint programming as the complexity of manipulating seeding mechanisms is unknown. The third problem concerned detecting manipulations of tournaments. A dynamic programming approach was proposed to identify manipulating coalitions in cup competitions. Additionally, the feasibility of detecting seeding manipulations was discussed.

In more detail, the first problem I addressed was extending previous work in qualification and elimination problems to the sport of hockey, specifically the qualification and elimination rules used in the National Hockey League. This work differed from previous work as it solves problems with more wild card teams and proposes a combination of network flows, enumeration and constraint programming to solve the problem. While network flows and enumeration have been used previously [63, 69], the specific tie breaking constraints used by National Hockey League were solved by the constraint programming model described in this work. The inclusion of tie breaking conditions into the problems was another area where little has been studied. Breaking problems into a phased approach proved to be an effective technique for solving tie breaking problems. Since tie breaking problems have a natural overlapping structure, it was useful to determine if the problem could be solved with only a single tie breaking problem before introducing additional tie breaking measures. Experimental data confirmed this hypothesis as a majority of the problems did not require the full tie breaking criteria. The results of the experiments showed that qualification could be shown several days earlier than the heuristic methods used by a major national newspaper. Problems could be solved efficiently and the precise number of points needed to clinch a playoff spot and the number of losses which could be absorbed and still have a chance of making the playoffs could be determined within seconds for most problems. Even the hardest instances could be solved in under ten minutes.

The second problem I addressed was manipulation in cup and round robin competitions, the two most common types of sports competitions. These two types of competition are used in sports at the local, national and international level. As such their susceptibility to manipulation is of interest. Work in elections and qualification problems was extended to problems of manipulating cups and round robins, respectively. Cup manipulations were found to have a polynomial algorithm for both constructive and destructive manipulation. Constructive round robin manipulations could be found in polynomial time given a specific class of manipulations and are NP-Hard to find otherwise. However, the polynomial class of manipulation contains the win-loss model and the model which awards two points for a win, one point for a tie and zero points for a loss. Since these models are among the most common, the susceptibility to constructive manipulation remains high. While the manipulation of tournaments using the scoring model of professional soccer are NP-Hard, in practice [54], these problems could be solved. A destructive round robin manipulation could always be found in polynomial time regardless of the scoring model. Since a coalition could potentially have a choice over which set of manipulations to choose, the question of manipulation strategy was pertinent. Since changing the result of the game increases the exposure of the coalition and potentially increases the cost, it is advantageous to manipulate the coalition using as few manipulations as possible. The work in Chapter 4 shows that finding these manipulations could be solved in polynomial time.

The complexity of seeding manipulation has been the focus of several different papers and still remains open [35, 28, 67, 70]. A recent paper showed that there exist several classes of problems where a polynomial algorithm exists. This work examined a different aspect of the problem. Given the unknown complexity of the problem, it experimentally evaluated the problem as is done with problems such as graph isomorphism. Additionally, the problem was expanded to look at restrictions of the seeding. The basic model allows for extremely unlikely seedings and in practice additional restrictions are placed on the schedulers of the tournament. These restrictions include a fixed pooling, a fixed arrangement of teams, a fixed arrangement of pool winners and a fixed ranking. Experimental evaluation showed that, regardless of the complexity, the simple model could be easily solved using constraint programming. This is unsurprising given the result by Williams which states that, in theory, it is likely that most problems have a solution given a random tournament graph [70]. Results in this work showed that, even with restrictions, most problems can be solved within seconds. Only in a small number of cases are there instances which require more time to solve.

The third problem I addressed was whether it was possible to detect the manipulations from the pattern of the results. Detecting manipulations given the result of a competition was a new research area. This work puts forth a method for finding coalitions in cup competition results. Using the notion of minimality from Chapter 4, the notion of a strategically optimal coalition was presented given a number of upsets. These strategically optimal coalitions were coalitions that react to upsets and always behave in a minimal manner in the next round. Using dynamic programming, it was possible to effectively detect strategically optimal coalitions. When comparing results without manipulating coalitions and those with manipulating coalitions, it was shown that the accuracy of detection is between 76% and 93% depending on the size of the competition. However, a large number of possible coalitions was returned when there was a coalition and it was not straight forward to determine which coalition to choose. However, experimental data showed that, in 95% of all instances the coalitions returned, nearly 60% of the teams were actually members of the coalition.

Detecting manipulations from round robins was broken into two categories. Those coalitions that manipulate by losing games to t_w and those games where top teams must lose games so that t_w can win. In both cases, it was shown that, for the instances tested, the accuracy of detection was better than 88% across both types of manipulations and all round robin sizes tested. One reason for this was that the specific behaviour exhibited by a manipulating team tends to be very different than a team that can be upset. It was also shown that on average the possible coalitions returned differed by less than one team. As well, for complex manipulations where top teams must lose, the number of extra possible coalitions returned is only 1.72 extra coalitions on average. One possible drawback is that more than one team may be identified as strategically optimal but this rarely happened in random instances and across all sizes was less than 37% of instances for a given size.

It was not always possible to detect manipulation from the seeding. The analysis of the different restrictions shows that when there was an expectation of fairness it was possible to calculate the divergence from an expected notion of fair. However, when a seeding feature was supposed to be generated by random process then it was not possible to claim evidence of manipulation as any random seeding was equally likely. For example, if the pools were selected by random draw then any random draw is valid and nothing distinguishes this draw from a manipulated draw. One reason for this is that as a single seeding there is no basis to judge bias. In the case of modification of the rank, it is possible to detect because there is an expectant outcome.

6.1 Future Work

For the qualification and elimination problems in NHL Hockey, two areas of future work exist. First, the solver structure developed in this work could be applied to other sports. One sport that seems to have been missed entirely is basketball, especially NBA basketball, where that league shares many similarities with the NHL. The tie breaking conditions vary but the NBA uses a simpler scoring model with only wins and losses. Second, the issue of the probability of elimination and qualification has not been addressed. In some cases, knowing how likely an event is to occur could be much more meaningful than the exact number of games needed to qualify or be eliminated. The probability of a team clinching a playoff spot or being eliminated from contention can be calculated by finding the sum of the probability of all scenarios where the distinguished team clinches or is eliminated. Note that as a consequence of solving this problem, the problem of counting all the situations where the team clinches or is eliminated must be solved, which is known to be $\#P$-Complete [27]. Another way to extend the work on qualification and elimination problems is to generate scenarios to determine if critical events need to happen for a team to clinch or escape elimination.

In the work on manipulation, a number of open problems and areas for future exists. One of the known open questions in this area of research is determining the complexity of seeding manipulation or schedule control for a balanced tournament. This problem has been studied in a variety of different papers [35, 28, 67, 70] but as yet only a small number of polynomial subclasses have been discovered [70]. Also, there are a number of variants of the standard cup and round robin competitions that have not been fully studied and their complexity is unknown. The open questions cover well known tournaments like the World Cup of Football. As well, some of the combinations of the different types of competitions and manipulations have unknown complexity. Further research in this area is needed.

In the work on detection, a few open questions and areas for future work have been identified. There are a number of restrictions added to the detection problem that are not true in every situation. To improve the applicability of these results, in the future, it would be interesting to evaluate the techniques proposed under less restrictive conclusions. Another open complexity question is whether it is possible to determine in polynomial time, given a competition, tournament graph and set of upsets, if there is a coalition which executed strategically optimal manipulations. One interesting question that arose from this research is in the design of competitions. Specifically, could it be possible to design, assuming forces external to the competition are removed, a reward method for a cup competition such that no coalition of teams is better off by manipulating the competition. This idea borrows from mechanism design the idea of an incentive compatible mechanism. These mechanisms may be impossible but that may also be an interesting result.

APPENDICES

Appendix A

NHL Qualification and Elimination Results

Table A.1: The results of qualification and elimination compared against the Globe and Mail published results in 2005-06 and 2006-07. Highlighting denotes that the team qualified for the playoffs. Note that the Globe and Mail did not publish elimination results in 2005-06.

	2005-06		2006-07	
Team	Optimal	Globe & Mail	Optimal	Globe & Mail
East				
Toronto	Apr 17	N/A	Apr 9	Apr 9
Ottawa	Mar 22	Mar 27	Mar 25	Mar 25 or 26
Montreal	Apr 14	Apr 18	Apr 8	Apr 8 or 9
Buffalo	Apr 4	Apr 5	Mar 18	Mar 22
Boston	Apr 2	N/A	Mar 30	Mar 31, Apr 1 or 2
NY Islanders	Apr 7	N/A	Apr 9	Apr 8 or 9
NY Rangers	Apr 5	Apr 5	Apr 6	Apr 6
New Jersey	Apr 12	Apr 14	Mar 28	Mar 28
Pittsburgh	Mar 21	N/A	Mar 28	Mar 28
Philadelphia	Apr 8	Apr 8	Mar 9	Mar 26
Washington	Mar 29	N/A	Mar 22	Mar 26
Tampa Bay	Apr 18	Apr 18	Apr 6	Apr 6
Florida	Apr 9	N/A	Apr 4	Apr 4
Atlanta	Apr 18	N/A	Apr 2	Apr 4
Carolina	Mar 28	Mar 28	Apr 4	Apr 4
West				
Vancouver	Apr 14	N/A	Mar 28	Mar 28
Edmonton	Mar 22	Mar 27	Mar 21	Mar 26
Calgary	Apr 14	Apr 18	Apr 8	Apr 8 or 9
Colorado	Apr 13	Apr 15	Apr 8	Apr 8 or 9
Minnesota	Apr 8	N/A	Mar 28	Mar 28
Los Angeles	Apr 14	N/A	Mar 13	Mar 26
Anaheim	Apr 11	Apr 12*	Mar 24	Mar 25 or 26
San Jose	Apr 13	Apr 14	Mar 28	Mar 29
Dallas	Mar 31	Apr 1, 2 or 3	Mar 28	Mar 28
Phoenix	Apr 7	N/A	Mar 16	Mar 26
Chicago	Mar 23	N/A	Mar 11	Mar 26
Detroit	Mar 27	Mar 28*	Mar 24	Mar 25 or 26
Columbus	Mar 23	N/A	Mar 18	Mar 26*
St. Louis	Mar 23	N/A	Mar 26	Mar 28
Nashville	Apr 9	Apr 10	Mar 23	Mar 24

Table A.2: Clinching and Elimination under different scoring models in the NHL for 2005-06 and 2006-07 seasons. Highlighting denotes that the team qualified for the playoffs.

	05-06					06-07				
Team	Historic	Overtime	Extra Point	Current	Proposed	Historic	Overtime	Extra Point	Current	Proposed
East										
Toronto	Apr 18	Apr 14	Apr 19	Apr 17	Apr 17	Apr 8	Apr 8	Apr 8	Apr 9	Apr 9
Ottawa	Mar 20	Mar 20	Mar 21	Mar 22	Mar 19	Mar 19	Mar 18	Mar 21	Mar 25	Mar 21
Montreal	Apr 14	Apr 16	Apr 12	Apr 14	Apr 14	Apr 8	Apr 8	Apr 8	Apr 8	Apr 7
Buffalo	Apr 4	Apr 6	Mar 31	Apr 4	Mar 30	Mar 24	Mar 21	Mar 23	Mar 18	Mar 22
Boston	Apr 12	Apr 6	Apr 7	Apr 2	Apr 4	Mar 23	Mar 26	Mar 26	Mar 30	Mar 29
NY Islanders	Apr 6	Apr 3	Apr 6	Apr 7	Apr 6	Apr 8	Apr 4	Apr 9	Apr 9	Apr 9
NY Rangers	Apr 4	Apr 19	Apr 7	Apr 5	Apr 5	Apr 8	Apr 4	Apr 9	Apr 6	Apr 7
New Jersey	Apr 10	Apr 14	Apr 17	Apr 12	Apr 14	Mar 25	Mar 16	Mar 30	Mar 28	Mar 28
Pittsburgh	Mar 27	Mar 25	Mar 23	Mar 21	Mar 22	Apr 1	Mar 30	Mar 30	Mar 28	Mar 30
Philadelphia	Apr 10	Apr 10	Apr 8	Apr 8	Apr 7	Mar 9	Mar 9	Mar 13	Mar 9	Mar 9
Washington	Apr 2	Mar 24	Mar 26	Mar 29	Mar 31	Mar 23	Mar 25	Mar 25	Mar 22	Mar 22
Tampa Bay	Apr 15	Apr 15	Apr 19	Apr 18	Apr 19	Apr 7	Apr 8	Apr 7	Apr 6	Apr 8
Florida	Apr 14	Apr 8	Apr 12	Apr 9	Apr 8	Apr 6	Apr 8	Apr 8	Apr 4	Apr 4
Atlanta	Apr 18	Apr 19	Apr 19	Apr 18	Apr 19	Apr 7	Apr 1	Apr 2	Apr 2	Apr 7
Carolina	Apr 1	Apr 4	Mar 28	Mar 28	Mar 24	Apr 8	Apr 2	Apr 6	Apr 4	Apr 8
West										
Vancouver	Apr 13	Apr 13	Apr 18	Apr 14	Apr 14	Apr 4	Mar 28	Mar 28	Mar 28	Apr 4
Edmonton	Apr 14	Apr 14	Apr 18	Apr 14	Apr 13	Mar 25	Mar 22	Mar 22	Mar 21	Mar 22
Calgary	Apr 6	Apr 6	Apr 6	Apr 8	Apr 6	Mar 26	Mar 28	Apr 1	Apr 8	Apr 1
Colorado	Apr 8	Apr 9	Apr 12	Apr 13	Apr 10	Apr 8	Mar 28	Apr 8	Apr 8	Apr 8
Minnesota	Apr 18	Apr 11	Apr 8	Apr 8	Apr 10	Apr 8	Mar 28	Apr 8	Mar 28	Apr 8
Los Angeles	Apr 16	Apr 14	Apr 14	Apr 14	Apr 14	Mar 25	Mar 14	Mar 16	Mar 13	Mar 16
Anaheim	Apr 7	Apr 9	Apr 10	Apr 11	Apr 10	Mar 17	Mar 24	Mar 24	Mar 24	Mar 28
San Jose	Apr 18	Apr 14	Apr 11	Apr 13	Apr 13	Mar 22	Mar 22	Mar 24	Mar 28	Mar 23
Dallas	Apr 1	Mar 31	Apr 7	Mar 31	Mar 30	Mar 28	Mar 25	Mar 31	Mar 28	Apr 1
Phoenix	Apr 10	Apr 11	Apr 9	Apr 7	Apr 9	Mar 19	Mar 16	Mar 16	Mar 16	Mar 18
Chicago	Apr 2	Mar 27	Mar 24	Mar 23	Mar 24	Mar 25	Mar 22	Mar 21	Mar 21	Mar 18
Detroit	Mar 24	Mar 24	Mar 24	Mar 27	Mar 20	Mar 14	Mar 15	Mar 15	Mar 24	Mar 15
Columbus	Mar 26	Mar 26	Mar 20	Mar 23	Mar 22	Mar 25	Mar 18	Mar 18	Mar 18	Mar 18
St. Louis	Mar 30	Mar 28	Mar 22	Mar 23	Mar 21	Apr 4	Mar 23	Mar 25	Mar 26	Mar 23
Nashville	Apr 2	Apr 2	Apr 9	Apr 9	Apr 7	Mar 18	Mar 12	Mar 24	Mar 23	Mar 18

Appendix B

Seeding Variation Results

Table B.1: The effect of tournament size on the minimum, maximum and average CPU time (sec.) for the families of restrictions. Results are averaged over 1000 instances.

	00XX		10X0		010X		10X1		1100		011X		1101	
size	range	avg	range	avg	range	avg	range	avg	range	avg	range	avg	range	avg
16	[0-1]	0.00	[0-1]	0.00	[0-1]	0.01	[0-1]	0.00	[0-1]	0.01	[0-0]	0.00	[0-1]	0.00
32	[0-1]	0.01	[0-1]	0.03	[0-8]	0.04	[0-1]	0.03	[0-1]	0.07	[0-1]	0.00	[0-1]	0.07
64	[0-1]	0.04	[0-1]	0.06	[0–1657]	1.3	[0-1]	0.06	[0-1]	0.12	[0-8]	0.04	[0-2]	0.12
128	[0-1]	0.27	[0-1]	0.13	[0–?][1]	N/A[1]	[0-2]	0.13	[0-6]	0.21	[0-61]	0.44	[0-3]	0.22
256	[1-61]	1.81	[0-1]	0.33	[0–?][2]	N/A[2]	[0-3]	0.29	[0-3]	0.43	[0-29]	3.22	[0-5]	0.42

[1] Due to memory restrictions, it was not possible to solve 9 out of the 12800 instances. As such, the maximum time and average cannot be accurately reported.

[2] Due to memory restrictions, it was not possible to solve 1794 out of the 25600 instances. As such, the maximum time and average cannot be accurately reported.

Table B.2: The effect of tournament size on the percentage of teams of rank $i = 1, \ldots, 16$, that could be made the winner via manipulation of the seeding, for all families of restrictions. Results are averaged over 1000 instances.

00XX	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
4	41	41	41	41	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
8	91	80	80	85	63	54	43	40	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
16	97	97	96	95	95	94	95	95	90	92	90	82	78	62	59	32
32	100	100	100	100	100	100	100	100	100	100	100	100	100	98	97	87
64	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
128	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
256	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
10X0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
16	97	97	96	95	95	94	95	95	90	92	90	82	78	62	59	32
32	97	96	95	96	94	93	94	93	91	89	87	85	76	67	48	37
64	99	98	99	99	98	98	98	98	95	96	92	88	78	64	52	40
128	99	97	97	97	97	96	96	96	94	92	91	85	75	66	55	44
256	98	98	97	97	97	97	96	96	94	93	90	87	77	65	56	39
010X	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
16	88	93	79	68	58	51	35	20	12	22	13	11	7	1	1	0
32	100	100	100	100	100	100	100	93	89	99	93	90	79	69	23	6
64	100	100	100	100	100	100	100	100	100	100	100	100	100	97	54	16
128	100	100	100	100	100	100	100	100	100	100	100	100	100	100	75[1]	32[1]
256	100	100	100	100	100	100	100	100	97[2]	100	100	100	96[2]	91[2]	52[2]	0[2]
10X1	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
16	97	97	96	95	95	94	95	95	90	92	90	82	78	62	59	32
32	97	96	95	96	94	93	94	93	91	89	87	85	76	67	48	37
64	99	98	98	98	97	97	97	96	94	94	90	85	75	63	50	37
128	97	95	95	95	94	93	92	93	90	88	88	80	71	62	51	39
256	97	96	96	96	95	94	93	93	91	89	86	83	74	61	52	34
1100	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
16	88	93	79	68	58	51	35	20	12	22	13	11	7	1	1	0
32	83	83	74	65	56	42	28	20	13	16	12	7	3	2	1	0
64	90	85	74	68	54	52	30	20	15	15	15	8	3	2	0	0
128	90	83	73	67	55	48	32	20	14	13	15	9	5	4	1	0
256	90	85	74	64	55	46	31	22	14	15	16	9	5	4	1	0
011X	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
16	19	26	26	17	5	3	1	0	0	1	1	1	0	0	0	0
32	86	84	69	66	44	29	21	20	8	7	6	4	2	2	0	0
64	100	100	100	100	98	94	84	77	46	60	60	57	43	33	13	0
128	100	100	100	100	100	100	99	93	79	96	91	93	82	69	32	0
256	100	100	100	100	100	100	100	100	95	100	100	100	96	90	52	0
1101	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
16	88	93	79	68	58	51	35	20	12	22	13	11	7	1	1	0
32	83	83	74	65	56	42	28	20	13	16	12	7	3	2	1	0
64	85	79	70	64	49	45	25	17	14	12	11	7	3	2	0	0
128	86	77	67	61	47	41	26	16	9	8	11	6	3	2	0	0
256	84	78	68	58	47	37	25	17	10	11	10	6	3	2	0	0

[1] Percentages are a lower bound of actual percentages with a difference from actual between 0% and 1%.

[2] Percentages are a lower bound of actual percentages with a difference from actual between 0% and 51%.

Appendix C

Round Robin Detection Results

Table C.1: The effect of round robin size on the accuracy (%) of identifying manipulation of round robin competitions by only manipulating games against t_w. 100 random and embedded instances are compared.

6	8	10	12	14	16	18	20	22
88.5	94.0	99.0	97.5	99.0	100.0	99.5	99.0	100.0

24	26	28	30	32	34	36	38	40
99.5	99.0	99.0	99.0	100.0	100.0	99.5	99.5	99.5

Table C.2: The effect of round robin size on the accuracy (%) of identifying manipulation of round robin competitions using complex manipulations where top teams must lose. 100 random and embedded instances are compared.

6	8	10	12	14	16	18	20	22
98.2	99.0	100.0	100.0	97.5	97.5	98.0	100.0	100.0

24	26	28	30	32	34	36	38	40
100.0	100.0	99.5	100.0	100.0	100.0	100.0	100.0	100.0

Table C.3: The effect of round robin size on the minimum, maximum and average difference between the expected coalition and $A \cup B$ where the coalition is a subset of $A \cup B$. Averages are over 100 instances.

6		8		10		12		14		16	
range	average	range	average	range	average	range	average	range	average	range	average
[0–2]	0.34	[0–2]	0.35	[0–2]	0.25	[0–1]	0.18	[0–2]	0.15	[0–2]	0.20

18		20		22		24		26		28	
range	average	range	average	range	average	range	average	range	average	range	average
[0–1]	0.16	[0–2]	0.26	[0–2]	0.12	[0–2]	0.19	[0–2]	0.13	[0–2]	0.15

30		32		34		36		38		40	
range	average	range	average	range	average	range	average	range	average	range	average
[0–3]	0.10	[0–2]	0.07	[0–2]	0.12	[0–2]	0.09	[0–3]	0.17	[0–1]	0.05

Table C.4: The effect of round robin size on the minimum, maximum and average difference between the generated coalitions and the expected coalition. Averages are over 100 instances.

6		8		10		12		14		16	
range	average	range	average	range	average	range	average	range	average	range	average
[0–2]	0.32	[0–2]	0.17	[0–4]	0.73	[0–2]	0.16	[0–4]	0.86	[0–3]	0.55

18		20		22		24		26		28	
range	average	range	average	range	average	range	average	range	average	range	average
[0–4]	0.48	[0–3]	0.25	[0–2]	0.54	[0–4]	0.64	[0–2]	0.49	[0–4]	1.55

30		32		34		36		38		40	
range	average	range	average	range	average	range	average	range	average	range	average
[0–3]	0.67	[0–5]	3.08	[0–2]	0.65	[0–3]	0.70	[0–3]	0.61	[0–3]	0.74

Table C.5: The effect of round robin size on the minimum, maximum and average number of generated coalitions for instances where a coalition was manipulating the competition. Averages are over 100 instances.

6		8		10		12		14		16	
range	average	range	average	range	average	range	average	range	average	range	average
[1–3]	1.24	[1–2]	1.05	[1–2]	1.05	[1–1]	1.00	[1–4]	1.44	[1–1]	1.00

18		20		22		24		26		28	
range	average	range	average	range	average	range	average	range	average	range	average
[1–2]	1.08	[1–3]	1.26	[1–14]	1.50	[1–21]	1.47	[1–24]	2.21	[1–4]	1.21

30		32		34		36		38		40	
range	average	range	average	range	average	range	average	range	average	range	average
[1–39]	1.89	[1–6]	1.47	[1–14]	1.73	[1–22]	2.72	[1–15]	1.59	[1–15]	2.18

Table C.6: The effect of round robin size on the percentage of instances with two or more teams identified as having won through strategically optimal coalitions using simple manipulations directly to t_w or complicated manipulations involving the top teams. Random instances are not shown as this occurred in only two simple random instances and no complex instances. 100 simple and 100 complex manipulations are considered.

Simple Manipulations						Complicated Manipulations					
6	8	10	12	14	16	6	8	10	12	14	16
37	32	25	27	27	29	4	6	10	12	19	17
18	20	22	24	26	28	18	20	22	24	26	28
25	23	22	25	24	24	20	15	16	17	11	15
30	32	34	36	38	40	30	32	34	36	38	40
24	18	23	29	20	28	17	12	15	11	9	16

References

[1] I. Adler, A. L. Erera, D. S. Hochbaum, and E. V. Olinick. Baseball, optimization and the world wide web. *Interfaces*, 32:12–22, 2002.

[2] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms and Applications*. Prentice Hall, 1993.

[3] A. Altman, A. D. Procaccia, and M. Tennenholtz. Nonmanipulable selections from a tournament. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence*, pages 27–32, 2009.

[4] D. H. Annis and B. A. Craig. Hybrid paired comparison analysis with applications to the ranking of college football teams. *Journal of Quantitative Analysis in Sports*, 1, 2005.

[5] Anonymous. Match-fix probe targets 200 games. BBC News, November 2009. http://news.bbc.co.uk/2/hi/europe/8370748.stm.

[6] E. Asinof. *1919: America's Loss of Innocence*. Dutton Adult, 1990.

[7] J. A. Aslam, R. A. Popa, and R. L. Rivest. On auditing when precincts have different sizes. In *Proceedings of the Conference on Electronic Voting Technology*, pages 1–13, 2008.

[8] R. Backofen. Using constraint programming for lattice protein folding. In *Proceedings of the 3rd Pacific Symposium on Biocomputing*, pages 387–398, 1998.

[9] A. N. Banerjee, J. F. M. Swinnen, and A. Weersink. Skating on thin ice: rule changes and team strategies in the NHL. *Canadian Journal of Economics*, 40:493–514, 2007.

[10] J. J. Bartholdi, C. A. Tovey, and M. A. Trick. The computational difficulty of manipulating an election. *Social Choice and Welfare*, 6:227–241, 1989.

[11] V. Boginski, S. Butenko, and P. M. Pardalos. Matrix-based methods for college football rankings. In S. Butenko, J. Gil-Lafuente, and P. M. Pardalos, editors, *Economics, Management and Optimization in Sports*, pages 1–14. Springer, 2004.

[12] E. F. Brickell and D. R. Stinson. The detection of cheaters in threshold schemes. In *Advances in Cryptology*, pages 564–577, 1990.

[13] J. R. Brown. The sharing problem. *Operations Research*, 27:324–340, 1979.

[14] T. Bryant. It's not just formula one ... match-rigging claims hit lawn bowls. The Guardian, September 2009. http://www.guardian.co.uk/sport/2009/sep/18/bowling-match-rigging-new-zealand.

[15] C. R. Cassady, L. M. Maillart, and S. Salman. Ranking sports teams: A customizable quadratic assignment approach. *Interfaces*, 35:497–510, 2005.

[16] E. Cheng and D. Steffy. Clinching and elimination of playoff berth in the NHL. *International Journal of Operations Research*, 5:187–192, 2008.

[17] B. J. Coleman, J. M. DuMond, and A. K. Lynch. Evidence of bias in NCAA tournament selection and seeding. *Managerial and Decision Economics*, 2010. Online March 2010.

[18] V. Conitzer, T. Sandholm, and J. Lang. When are elections with few candidates hard to manipulate? *Journal of the ACM*, 54:1–33, 2007.

[19] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 2001.

[20] D. de Werra. Scheduling in sports. *Annals of Discrete Mathematics*, 11:381–395, 1981.

[21] M. Duggan and S. D. Levitt. Winning isn't everything: Corruption in sumo wrestling. *The American Economic Review*, 92:1594–1605, 2002.

[22] P. Faliszewski, E. Hemaspaandra, and H. Schnoor. Copeland voting: Ties matter. In *Proceedings of the 7th International Conference on Autonomous Agents and Multiagent Systems*, pages 983–990, 2008.

[23] G. Garber. Gamesmanship is name of the game in tennis. ESPN.com, August 2007. http://sports.espn.go.com/espn/cheat/news/story?id=2955743.

[24] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979.

[25] O. Gibson. UEFA investigates 40 European games in match-fixing crackdown. The Guardian, September 2009. http://www.guardian.co.uk/football/2009/sep/25/uefa-match-fixing-champions-league.

[26] Globe and Mail. Hockey scoreboard. Metro ed., 2006-2007.

[27] D. Gusfield and C. E. Martel. The structure and complexity of sports elimination numbers. *Algorithmica*, 32:73–86, 2002.

[28] N. Hazon, P. E. Dunne, S. Kraus, and M. Wooldridge. How to rig elections and competitions. In *Proceedings of the 2nd International Workshop on Computational Social Choice*, pages 301–312, 2008.

[29] P. Van Hentenryck. *Constraint Satisfaction in Logic Programming.* MIT Press, 1989.

[30] A. Huang. Say it ain't so Tsao: Baseball scandal hits Taiwan. The Associated Press, October 2009.

[31] ILOG S.A. ILOG Solver 4.2 user's manual, 1998.

[32] H. Jin, J. Lotspiech, and N. Megiddo. Efficient coalition detection in traitor tracing. In *Proceedings of the 23rd International Information Security Conference*, pages 365–380, 2008.

[33] U. Junker. Preference programming for configuration. In *Proceedings of the 4th Workshop on Configuration*, pages 50–56, 2001.

[34] W. Kern and D. Paulusma. The computational complexity of the elimination problem in generalized sports competitions. *Discrete Optimization*, 1:205–214, 2004.

[35] J. Lang, M. S. Pini, F. Rossi, K. B. Venable, and T. Walsh. Winner determination in sequential majority voting. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, pages 1372–1377, 2007.

[36] J. Larrosa and G. Valiente. Constraint satisfaction algorithms for graph pattern matching. *Mathematical Structures in Computer Science*, 12:403–422, 2002.

[37] D. Lesaint, D. Metha, B. O'Sullivan, L. Quesada, and N. Wilson. Solving a telecommunications feature subscription configuration problem. In *Proceedings of the 14th International Conference on the Principles and Practice of Constraint Programming*, pages 67–81, 2008.

[38] I. Levin, G. A. Cohn, P. C. Ordeshook, and R. M. Alvarez. Detecting voter fraud in an electronic voting context: An analysis of the unlimited reelection vote in Venezuela. In *Proceedings of the 2009 Electronic Voting Technology Workshop/Workshop on Trustworthy Computing*, pages 1–23, 2009.

[39] O. Lhomme. Arc-consistency filtering algorithms for logical combinations of constraints. In *Proceedings of the 10th International Conference on the Principles and Practice of Constraint Programming*, pages 209–224, 2004.

[40] M. Maher, N. Narodytska, C.-G. Quimper, and T. Walsh. Flow-based propagators for the sequence and related global constraints. In *Proceedings of the 14th International Conference on Principles and Practice of Constraint Programming*, pages 159–174, 2008.

[41] A. Malik, M. Chase, T. Russell, and P. van Beek. An application of constraint programming to superblock instruction scheduling. In *Proceedings of the 14th International Conference on the Principles and Practice of Constraint Programming*, pages 97–111, 2008.

[42] C. D. Manning and H. Shutze. *Foundations of Statistical Language Processing.* MIT Press, 1999.

[43] K. Marriott and P. Stuckey. *Programming with Constraints: An Introduction.* MIT Press, 1998.

[44] J. Martinich. College football rankings: Do the computers know best? *Interfaces*, 32:85–94, 2002.

[45] S. T. McCormick. Fast algorithms for parametric scheduling come from extensions to parametric maximum flow. *Operations Research*, 47:744–756, 1999.

[46] A.V. Moura, C.C. de Souza, A.A. Cire, and T. M. T. Lopes. Planning and scheduling the operation of a very large oil pipeline network. In *Proceedings of the 14th International Conference on the Principles and Practice of Constraint Programming*, pages 36–51, 2008.

[47] M. Myagkov, P. C. Ordeshook, and D. Shakin. *The Forensics of Election Fraud: Russia and Ukraine.* Cambridge University Press, 2009.

[48] NCAA. NCAA division I men's basketball championship: Principles and procedures for establishing the bracket. http://www.ncaa.com/graphics/champpage/Bracket_Prin-Proc_2009-10_07.07.09.pdf, 2009.

[49] G. Nemhauser and M. Trick. Scheduling a major college basketball conference. *Operations Research*, 26(1):1–8, 1998.

[50] M. J. Osborne and A. Rubenstein. *A Course in Game Theory.* MIT Press, 1994.

[51] G. Pesant, M. Gendreau, J. Y. Potvin, and J. M. Rousseau. An exact constraint logic programming algorithm for the traveling salesman problem with time windows. *Transportation Science*, 32:12–29, 1998.

[52] M.S. Pini, F. Rossi, K. B. Venable, and T. Walsh. Dealing with incomplete agents' preferences and an uncertain agenda in group decision making via sequential majority voting. In *Proceedings of the 11th International Conference on Principles of Knowledge Representation and Reasoning*, pages 571–578, 2008.

[53] T. Rahwan, S. D. Ramchurra, N. R. Jennings, and A. Giovannucci. An anytime algorithm for optimal coalition structure generation. *Journal of Artificial Intelligence Research*, 34:523–567, 2009.

[54] C. C. Ribeiro and S. Urrutia. An application of integer programming to playoff elimination in football championships. *International Transactions in Operational Research*, 12:375–386, 2005.

[55] L. W. Robinson. Baseball playoff eliminations: an application of linear programming. *Operations Research Letters*, 10:67–74, 1991.

[56] F. Rossi, P. van Beek, and T. Walsh, editors. *The Handbook of Constraint Programming.* Elsevier, 2006.

[57] T. Russell and P. van Beek. Mathematically clinching a playoff spot in the NHL and the effect of scoring systems. In *Proceedings of the 21st Conference of the Canadian Society for Computational Studies of Intelligence*, pages 234–245, 2008.

[58] T. Russell and P. van Beek. Determining the number of games needed to guarantee an NHL playoff spot. In *Proceedings of the 6th International Conference on the Integration of AI and OR Techniques in CP for Combinatorial Optimization Problems*, pages 233–247, 2009.

[59] T. Russell and P. van Beek. Lessons learned from modelling the NHL qualification problem. In *Proceedings of the Eighth International Workshop on Constraint Modelling and Reformulation*, pages 132–146, 2009.

[60] T. Russell and T. Walsh. Manipulating tournaments in cup and round robin competitions. In *Proceedings of the First International Conference on Algorithmic Decision Theory*, pages 26–37, 2009.

[61] R. Rymon. Search through systematic set enumeration. In *Proceedings of the Third International Conference on the Principles of Knowledge Representation and Reasoning*, pages 268–275, 1992.

[62] R. Saltzman and R. M. Bradford. Optimal realignments of the teams in the national football league. *European Journal of Operations Research*, 93:469–475, 1996.

[63] B. Schwartz. Possible winners in partially completed tournaments. *SIAM Review*, 8:302–308, 1966.

[64] J. Siek, L.-Q. Lee, and A. Lumsdaine. *Boost Graph Library: User Guide and Reference Manual.* Addison-Wesley, 2001.

[65] S. Sorlin and C. Solnon. A global constraint for graph isomorphism problems. In *First International Conference on the Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 287–302, 2004.

[66] P. Tang, Y. Shoham, and F. Lin. Team competitions. In *Proceedings of 8th International Conference on Autonomous Agents and Multiagent Systems*, pages 241–248, 2009.

[67] T. Vu, A. Altman, and Y. Shoham. On the complexity of schedule control problems for knockout tournaments. In *Proceedings of 8th International Conference on Autonomous Agents and Multiagent Systems*, pages 225–232, 2009.

[68] T. Walsh. Where are the really hard manipulation problems? the phase transition in manipulating the veto rule. In *Proceedings of 21st International Joint Conference on Artificial Intelligence*, pages 324–329, 2009.

[69] K. D. Wayne. A new property and a faster algorithm for baseball elimination. *SIAM Journal on Discrete Mathematics*, 14:223–229, 2001.

[70] V. V. Williams. Fixing a tournament. In *Proceedings of the 24th AAAI Conference on Artificial Intelligence*, pages 895–900, 2010.

[71] J. Yan. Security design in online games. In *Proceedings of the 19th Annual Computer Security Applications Conference*, pages 286–295, 2003.

[72] S. Zampelli, Y. Deville, and C. Solnon. Solving subgraph isomorphism problems with constraint programming. *Constraints*, 15:327–353, 2010.