On Consulting a Set of Experts and Searching

by

Igal Galperin

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 1996

Author ....
Department of Electrical Engineering and Computer Science
September 5, 1996

Certified by ....
Ronald L. Rivest
E.S. Webster Professor of Computer Science
Thesis Supervisor

Accepted by ....
Frederic R. Morgenthaler
Chairman, Departmental Committee on Graduate Students

On Consulting a Set of Experts and Searching

by

Igal Galperin

Submitted to the Department of Electrical Engineering and Computer Science
on September 5, 1996, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

Two chapters of this thesis analyze expert consulting problems via game theoretic models; the first points out a close connection between the problem of consulting a set of experts and the problem of searching. The last chapter presents a solution to the dictionary problem of supporting SEARCH and update (INSERT and DELETE) operations on a set of key values.

The first chapter shows that the problem of consulting experts on-line can be modeled by a chip game similar and in some cases identical to the Paul-Carole games used to model a faulty search process. It presents the best known worst-case algorithms for consulting finitely many experts, and the best possible algorithms for consulting infinitely many experts (model selection) under some assumptions. It includes new results about faulty search processes as well as generalizations and new proofs of some known results.

The second chapter uses properties of coalitional games to analyze the performance of the greedy heuristic for the problem of hiring experts from a pool of candidates using stochastic data. The results are instrumental in suggesting an alternative to a known algorithm for learning Lipschitz functions by a memory-based learning systems via an analysis of the greedy approximate solution of the s-median problem.

The third and last chapter is dedicated to the *Scapegoat* trees data structure: a solution to the dictionary problem that uses binary trees with no auxiliary balancing data stored at the tree nodes to achieve logarithmic worst-case search time, and logarithmic amortized update time.

All chapters explore alternatives to the now standard worst-case analysis of algorithms. The first chapter introduces and advocates the notions of opportunism and almost opportunism of on-line algorithms. The second chapter contrasts the pessimism of worst-case analysis with the optimism of the greedy heuristic, and points out some benefits of exploring the latter. The last chapter evaluates a novel data structure by computing its amortized performance.

Thesis Supervisor: Ronald L. Rivest
Title: E.S. Webster Professor of Computer Science

Acknowledgments

I would like to thank the thesis supervisor Ron Rivest. He proposed the topics this thesis addresses, and posed questions that advanced my research. The details owe much to his careful reading and patient comments.

The thesis committee members John Tsitsiklis and David Karger made helpful suggestions.

Javed Aslam, Shai Ben-David, Michael Ben-Or, Scott Decatur, Peter Elias, Charles Leiserson, Natti Linial, Gabor Lugosi, Rafail Ostrovsky, Jeffrey Uhlmann, and anonymous conference committee members listened/read and provided advice. I also thank the support stuff of the theory group, in particular Be Hubbard, as well as many of its members throughout my stay here.

I thank the many people who showed me generosity, tolerance, sympathy, encouragement and support. I believe that without the extraordinary efforts of my family – parents and sister – the completion of this work would not be possible.

I thank all those who helped yet were not thanked so far.

Contents

List of Figures

Chapter 0

Introduction

This thesis addresses two problems – consulting a set of experts and searching. The problem of consulting a set of experts, or combining information from different sources to reach conclusions, is a most commonly-occurring problem. Many algorithmic tasks of producing a specific output based on a given input fit this description.

Many, if not all human behaviors seem to be the result of processing incoming information from multiple sources. Turing's test suggests intelligence can be measured by its resemblance to human behavior. The first two chapters of this thesis evolved from research in the area of Computational Learning Theory [38], the area of theoretical computer science that is most closely related to research on artificial intelligence.

We do not address the problem of consulting experts in its full generality. Rather we are concerned with two aspects of it: consulting experts on-line under worst-case assumptions about the input and selecting a "good" subset from a given pool of experts.

The problem of searching for a particular named element within a given set is one of the fundamental problems of theoretical computer science [49]. This thesis (in chapter 1) establishes a close connection between the problems of searching a set using unreliable information, and consulting experts on-line.

The last chapter addresses another variant of the problem of searching, that of finding an element in a dynamically changing set.

0.1 Chapter 1: Consulting a Set of Experts On-Line and Faulty Searching

The first chapter defines and explores a class of multistage games that capture information theoretic aspects of on-line learning. Our framework encompasses both the problem of consulting finitely many experts and the problem of model selection from infinitely many candidates. We introduce the PM algorithms which achieve the game's value for some families of inputs and come within a constant multiplicative factor for others. Thus they provide simultaneous upper and lower bounds on the complexity of the problems addressed.

Worst-case analysis of algorithms can be justified as a search for a solution that minimizes the risk involved. However, in applications the algorithm will often be faced with "easier" than the worst-case inputs. By introducing new notions of algorithmic complexity, *opportunism* and *almost opportunism*, we distinguish on-line algorithms which take full advantage of favorable situations, without unnecessary risks.

Using this new notion of opportunism, we prove that our algorithms, unlike previous algorithms, are almost opportunistic in games with finitely many experts. This suggests that to achieve optimal on-line learning performance the manager has to gather information in rounds in which she does not err, as well as in rounds in which she errs. This conclusion is in contrast to the often indistinguishable asymptotic performance of learning algorithms that gather information in all rounds and those that gather it only in rounds in which the algorithm errs.

We apply the PM algorithms to the previously unaddressed question of consulting experts over arbitrary finite decision domains of size ≥ 2, and also allow the learner to incorporate a prior on experts' quality.

The family of games discussed herein is closely related to the well-investigated Paul-Carole search games. In these games a searcher, Paul, tries to find a target value from a set of candidate values by questioning Carole. Carole is allowed to lie in some of her answers. It is shown that games in which the manager is evaluated on the number of mistakes he makes are reducible to games similar to the standard Paul-Carole search games in which the goals of the two sides are reversed, while expert consulting games in which the manager is evaluated on the number of mistakes he makes in excess of his best advisor or advisors are reducible to the standard Paul-Carole search games. Our analysis of these games allows a uniform derivation of generalizations of some known results.

For decision makers our proof offers some insight into the folk wisdom asser that "the hardest decisions to make are the least important ones".

The algorithms presented are named after a combinatorial entity they utilize, the Pascal Matrix.

0.2 Chapter 2: Greedy Expert Hiring and an Application

The second chapter (based on Galperin [27]) addresses the problem of hiring a set of experts from a pool of candidates. Modeling this problem by a coalitional game, a uniform lower bound on the performance of the greedy heuristic for a family of games is derived. We show a uniform bound for this family also holds when only approximate rather than exact values of coalitions are known. One of the prettiest applications of this general analysis is to the s-median problem.

Approximation algorithms for the s-median problem are a useful tool in learning Lipschitz functions in the generalized PAC learning model of Haussler [34, 35]. To approximate a Lipschitz function a memory-based learning system can be used, as proposed by Lin and Vitter [42]. We generalize the analysis of a greedy approximate solution of the s-median problem first considered by Cornuejols et al. [21]. We then compare its performance to the performance of Lin and Vitter's linear programming approximate solution of the same problem as a tool in the construction of memory-based learning systems. We find the greedy approximation is simpler, more efficient and in many cases yields a smaller system.

0.3 Chapter 3: Searching a Dynamically Changing Set

The last chapter (based on Galperin and Rivest [28]) is dedicated to the problem of supporting searches of a dynamically changing set of keys. An algorithm for maintaining binary search trees is presented. The amortized complexity per INSERT or DELETE is $O(\log n)$ while the *worst-case* cost of a SEARCH is $O(\log n)$.

Scapegoat trees, unlike most balanced-tree schemes, do not require keeping extra data (e.g. "colors" or "weights") in the tree nodes. Each node in the tree contains only a key value and pointers to its two children. Associated with the root of the whole tree are the only two extra values needed by the scapegoat scheme: the number of nodes in the whole tree, and the maximum number of nodes in the tree since the tree was last completely rebuilt.

In a scapegoat tree a typical rebalancing operation begins at a leaf, and successively examines higher ancestors until a node – the *scapegoat* – is found that is so unbalanced that the entire subtree rooted at the scapegoat can be rebuilt at zero cost, in an amortized sense.

Like the algorithms of the previous section, the algorithm for maintaining scapegoat trees enjoys the benefit of simplicity.

Chapter 1

Opportunistic Algorithms for Expert Advisees

1.1 Introduction

The problem of consulting a set of experts is a "master problem" that encompasses many other problems. In some cases we can ask how do humans generate their behavior based on sensory input, or how can a computer produce a desired output as a function of the boolean "advice" given by its input bits.

Here we explore the problem of consulting on-line a set of experts providing boolean advice to a manager who has to reach a boolean decision. It can be described as follows: A manager and a set of experts are presented a common "yes/no" question. The experts advise the manager on the correct reply, then she makes a decision, after which the correct reply is revealed. We explore the worst-case performance of the manager, and hence assume that the experts' votes as well as the correct reply are chosen by an adversary. This is repeated in a sequence of rounds. The manager's aim is to stay not too far behind the best advisors, when all are evaluated by the number of mistakes they made on the sequence. More cannot be hoped for in the worst-case, as no *a priori* assumptions are made about the experts or about the problem domain.

The problem of algorithmically consulting a finite set of experts on-line has been investigated under a variety of assumptions. (The on-line problem is distinguished by the manager having to reach a decision in every round, before seeing all inputs.) Littlestone and Warmuth's [44] WM (Weighted Majority) algorithm and later the BW (Binomial Weighting) algorithm by Cesa-Bianchi et al. [18] address the question of consulting a finite number of experts that provide boolean advice. The prediction domain of either the manager or the experts may be modified to be the real interval $[0, 1]$ as in Cesa-Bianchi et al. [17], Littlestone and Warmuth [44], Haussler et al. [32] and Vovk [70]. For this prediction domain various loss functions may be considered (Vovk [70], Haussler et al. [32]).

We generalize the problem of consulting finitely many experts by considering arbitrary measurable sets of experts of finite measure, and letting the decision domain of the manager and her advisors be a finite set. We discuss optimal algorithms in the worst-case against a computationally unlimited adversary. This is the set-up commonly assumed in the analysis of algorithms for finitely many consultants. The PM (Pascal Matrix) algorithms can be seen as a generalization of the Halving algorithm (Angluin [4], Barzdin and Freivalds [8]) to the case when the candidate predictors are allowed multiple errors.

A zero-sum multi-stage game is a competition between two players. Popular examples include chess, checkers, backgammon. An on-line algorithm, one performing a multi-round "conversation" with the user (as opposed to an off-line algorithm that may be seen as answering unrelated questions) may be looked upon as the computer's strategy in a multi-stage game between the computer and its user, in which the computer is challenged to meet some performance criterion measured by the Cost function in the following. The optimal worst-case strategy σ for the computer is one which for every state S that may be reached in the game and for every input sequence I incurs a cost no greater than

$$\inf_{\sigma \in \text{Strategies}} \sup_{I \in \text{Inputs}} \text{Cost}[\sigma(S, I)]. \tag{1.1}$$

The notion of state may be seen as capturing the relevant information about the history of an interaction between the computer and its user, as well as information known to the computer that is not part of this interaction. The reader is referred to Section 1.2 for formal definitions of the game theoretic terms. The $\inf \sup \text{Cost}[\sigma(S, I)]$ might not be a $\min \sup \text{Cost}[\sigma(S, I)]$ – the latter might be unachievable. If the measure of performance is not the running time of the algorithm, the $\min \sup$ strategy might not be computationally efficient, and an approximation might be necessary. Finding a $\min \sup$ algorithm within a certain class of algorithms assures us this is the best possible worst-case algorithm in the class. An algorithm which is the $\min \sup$ of all computationally unlimited randomized strategies we call an *opportunistic* algorithm.

Consider the problem of analyzing the performance of health care procedures (e.g. drug administration) or alternatively stock market investment policies. The worst-case analysis of algorithms is based on often unrealistic "pessimistic" assumptions yet it may be well justified under such circumstances as a means of risk minimization. As the medical treatment proceeds, however, we would like the treatment algorithm to take advantage of developments which will typically be more favorable than the worst-case development the treatment plan is assuming a priori. The notions of opportunism and almost-opportunism can facilitate the design of such algorithms.

An algorithm $\mathcal{A}$ is said to be K *almost opportunistic* or $\langle K, K' \rangle$ *almost opportunistic* if the cost of its execution starting with any reachable state S satisfies

$$\sup_{I \in \text{Inputs}} \text{Cost}\,\mathcal{A}(S, I) \leq K \inf_{\sigma \in \text{Strategies}} \sup_{I \in \text{Inputs}} \text{Cost}[\sigma(S, I)] + K'$$

for all possible inputs. We derive an efficiently solvable equation that gives the value of states in some expert consulting games and upper bounds this value in other games. Using this formula, we formulate algorithms which are opportunistic for some game classes and almost opportunistic for others.

The research efforts of theoreticians are dedicated to generating and evaluating algorithmic solutions. The most commonly used measure of algorithms' quality is their asymptotic performance. Yet this measure is not sensitive enough to separate the LRU and FIFO paging algorithms from LFU and LIFO although they are not equivalent in practice. Sleator and Tarjan [65] used competitivity to better compare them theoretically. The competitive ratio of an on-line algorithm $\mathcal{A}$ is said to be K if constant K, K' exist for which

$$\sup_{I \in \text{Inputs}} \text{Cost}\,\mathcal{A}(S_0, I) \leq K \inf_{\sigma \in \text{Off-strategies}} \sup_{I \in \text{Inputs}} \text{Cost}[\sigma(S_0, I)] + K'$$

where the infimum is taken over some set of off-line strategies, and the comparison of performance is carried out only for the initial state S_0 at which the game begins. While PM is 2 almost opportunistic for games with finitely many experts the best algorithm known currently for this problem, BW, is not. However, the bounds on their asymptotic as well as competitive performance are identical. Thus, almost opportunism is a complexity measure that in some cases is more sensitive than other known measures of algorithms' quality.

One of the main motivations for the exploration of lower bounds lies in the fact that through establishing lower bounds for algorithmic problems we find out how far known solutions for a certain problem are from its "optimal" solution. Opportunism and competitivity are more precise indicators of this "closeness" than the commonly used comparison of asymptotic lower and upper bounds since they do not neglect constants. However, reaching a competitive ratio of one for a problem is an unrealistic goal in most cases, as familiarity with future inputs seems to be indispensable information for the optimization of an algorithm. Proving an algorithm is opportunistic within some set of algorithms is the most realistic yet precise method at our disposal of stating an on-line algorithm cannot be improved upon in the worst-case.

The superiority of the PM algorithms offers evidence in favor of updating expert weights in every round of the game, rather than only in rounds in which the manager errs. We prove the PM algorithms achieve similar bounds when the decision domain is a finite set of arbitrary size. This generalizes the problem previously addressed by Littlestone and Warmuth [44] and Cesa-Bianchi et al. [18] of decision domains of size two ("yes/no" questions). The PM algorithms can also be applied to tracing "good" sets of experts of arbitrary size ≥ 1, if such are known to exist, and allow the manager to incorporate a non-uniform prior on experts' quality.

Expert Games explored herein are closely related to, and for some variants reducible to, the well investigated (Rivest et al. [61], Pelc [60], Aslam and Dhagat [6], Spencer and Winkler [67], Aslam [5]) Paul-Carole chip games modeling a faulty search process. Our analysis leads to an extension of results by Rivest et al. [61] to cover searchers that incorporate an arbitrary prior on the values searched. It yields the value of the continuous Mistake Bound game (Version A in Spencer and Winkler's [67]), implying a lower bound on the performance of Paul in the discrete game. It yields the same necessary and a different sufficient condition for Paul's victory than those specified by Spencer [66].

The heart of our proof method consists of establishing that the expert consulting problem can be modeled by a chip game (see e.g. Aslam and Dhagat [6]) in which the chooser tries to lengthen the game. We proceed to prove strong results about non-atomic expert consulting chip games. These imply somewhat weaker results for the interesting class of discrete games – atomic games that represent consulting finitely many experts. Our results are similar to those established by Cesa-Bianchi et al. [18] for consulting finitely many experts and those by e.g. Spencer [66] for Paul-Carole games, showing the close relationship between these two problems.

It is said that "the hardest decisions in life are the least important ones". This principle is accounted for by the observation that a decision is difficult to make when the options presented are of similar utility. Our proof suggests that such difficult to make decisions are hard in yet another way. It points out that in when the options are hard to differentiate the decision maker may be facing an adversarial choice.

Section 1.2 includes a few definitions and conventions. Section 1.3 begins with some facts about multi stage games, and moves on to define expert consulting games. In Section 1.4 we make observations about the adversary's optimal strategies. Then Section 1.5 uses these to derive the values of some games. Knowing these values allows a formulation of algorithms for the expert manager. Section 1.6 discusses the efficient implementation of these algorithms, and their absolute performance. Section 1.7 compares the PM algorithms to the already known BW and WM algorithms for the same problem and extends them to decision domains of size ≥ 2. It also addresses the implications of our results for the analysis faulty search processes.

1.2 Definitions, Notations, Conventions

Notation 1.2.1 *Denote by*

$$\mathrm{a} \gg \mathrm{n}$$

the result of shifting vector a right n positions and shifting in n zeros on the left. E.g.

$$\langle 0, 0, 1, 2, 3 \rangle \gg 2 = \langle 0, 0, 0, 0, 1, 2, 3 \rangle.$$

Denote by

$$\mathrm{a} \gg_{\mathrm{LP}} \mathrm{n}$$

the result of shifting vector a right while preserving its length. E.g.

$$\langle 0, 0, 1, 2, 3 \rangle \gg_{LP} 2 = \langle 0, 0, 0, 0, 1 \rangle.$$

Occasionally, as implied by the context, we may use $\gg$ to refer to shifts that preserve *vectors' lengths.*

Similarly $\ll$, $\ll_{LP}$ *denote left shifts.*

The shift operation can be applied to matrices as well, row-wise. When a matrix is shifted the reader should assume its size is not changed unless specified otherwise explicitly. E.g.

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 5 & 0 \\ 1 & 2 & 1 & 0 \\ 0 & 1 & 2 & 1 \end{pmatrix} \ll 1 = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 5 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 1 & 2 & 1 & 0 \end{pmatrix}.$$

When we multiply matrices by vectors we use the convention that when a matrix is multiplied by a row vector on the right the vector should be transposed

$$\begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 1 & 2 & 1 \end{pmatrix} \cdot \langle 0, 1, 2 \rangle = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 1 & 2 & 1 \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 1 \\ 2 \end{pmatrix}.$$

The convention for priority of operators is : for vectors a, b and a scalar α

$$b + \alpha a \gg 1 = b + ((\alpha a) \gg 1).$$

Notation 1.2.2 *For two vectors a, b of the same length, if all the coordinates of vector a are greater or equal to the corresponding coordinates of vector b, i.e. $\forall i \,.\, a_i \geq b_i$, then this is denoted*

$$\mathrm{a} \geq \mathrm{b}.$$

Definition 1.2.1 *A measure π on a non-empty set X is called* non-atomic *if for every set $T \subseteq X$ and every positive real number $c \leq 1$ there exists a set $T' \subseteq T$ such that $\pi(T') = c\pi(T)$. Otherwise, π is called* atomic.

Notation 1.2.3

$$\binom{m}{\leq k} = \sum_{i=0}^{k} \binom{m}{i}.$$

Notation 1.2.4

$$\mathbf{1}_j = \langle \underbrace{1, \ldots, 1}_{j}, 0, \ldots, 0 \rangle.$$

The dimension of $\mathbf{1}_j$ is stated or implied by the context when this notation is used.

For a set X we denote $X^* = \{f : \mathcal{N} \to X\}$, $\mathcal{N}$ denoting the natural numbers; and denote $\Re^+ = [0, \infty)$.

Definition 1.2.2 *The* 1-shift *is a relation on* $(\Re^+)^*$*, that we denote by* $\geq_{\mathrm{S}1}$*. For two vectors* $a, b \in (\Re^+)^*$ *the relation* $a \geq_{S1} b$ *holds iff there exist vectors* $y_1, y_2 \in (\Re^+)^*$ *such that*

$$\begin{aligned} y_1 &+ y_2 = a \\ y_1 &+ y_2 \gg 1 = b. \end{aligned}$$

Definition 1.2.3 *The* shift order *is a partial order on* $(\Re^+)^*$ *denoted* $\geq_{\mathrm{S}}$*. For two vectors* $a, b \in (\Re^+)^*$ *the relation* $a \geq_S b$ *holds iff there exist* $m \geq 0$ *vectors* $y_1, \ldots, y_m \in (\Re^+)^*$ *such that*

$$a \geq_{S1} y_1 \geq_{S1} \ldots \geq_{S1} y_m \geq b.$$

The shift order is the transitive closure of the 1-shift relation.

Spencer and Winkler [67] use an alternative equivalent definition. For two vectors $a, b \in (\Re^+)^*$, the relation $a \geq_S b$ holds iff for all i

$$\sum_0^i a_j \geq \sum_0^i b_j.$$

(Add trailing zeroes as needed.)

1.3 The Math of the Game

This section quotes some definitions and facts from Game Theory. It defines the notions of an opportunistic and almost opportunistic strategies. Then it describes the particulars of expert consulting games and defines them formally. These are the subject of discussion in what follows.

1.3.1 Multistage Two-Person Games

A zero-sum multistage two-person game [56] is a seven-tuple $G = \langle \mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_T, S_0, c, M_1, M_2 \rangle$. The set $\mathcal{S} = \mathcal{S}_1 \cup \mathcal{S}_2 \cup \mathcal{S}_T$ is the set of *states* of game G. The set $\mathcal{S}_T$ specifies the *terminal states*, those at which the game terminates. Other states we call *non-terminal*. State $S_0 \in \mathcal{S}_1 \cup \mathcal{S}_2$ is the game's *starting state*. Depending on whether $S_0 \in \mathcal{S}_1$ or $S_0 \in \mathcal{S}_2$ either the first or second player makes the first move. The value function $c : \mathcal{S}_T \rightarrow \Re$ determines the *value* of terminal states for the first player. Intuitively, this is the "sum" the second player pays the first when a game ends. We often look at games from the second player's perspective calling the same function the second player's *cost*, hence the c notation. We may also refer to it as specifying the players' *payoffs*. The *legal moves* of the respective players in each state are specified by the functions $M_i : \mathcal{S}_i \rightarrow \mathbf{2}^{\mathcal{S}_{3-i} \cup \mathcal{S}_T} / \{\emptyset\}$. We denote by $\mathbf{2}^{\mathcal{T}}$ the set of subsets of set $\mathcal{T}$. We only refer to zero-sum two-person games which we may simply call multistage games in the following.

A game starts at state S_0 and proceeds in rounds. In each round one of the players alternately makes a move which modifies the game's state. A move is selected by a player from the set of legal moves available to him that is specified by his M_i function. It depends on the game's state at the beginning of the round.

A state is *reachable* if some game play arrives at that state. A *pure strategy* for a player specifies a legal move for each state reachable from the start state S_0. Formally, we may define a reachable-set–strategy pair $\langle \mathcal{S}_{\sigma^1}, \sigma^1 \rangle$, where $\sigma^1 : \mathcal{S}_{\sigma^1} \rightarrow M_1(s)$ is a strategy, and $\mathcal{S}_{\sigma^1}$ is the set of states that may be reached when the first player plays σ^1 against some (at least one) strategy of the second player. A *mixed strategy* specifies a probability distribution over legal moves for all reachable states.

For, possibly mixed, strategies σ_1, σ_2 of the respective players denote by $\mathcal{V}(G|\sigma_1, \sigma_2)$ the expected payoffs for the players when these strategies are played. The *first player's value* *of game G with respect to* Σ_1, Σ_2 is

$$\mathcal{V}^1(G|\Sigma_1, \Sigma_2) = \sup_{\sigma_1 \in \Sigma_1} \inf_{\sigma_2 \in \Sigma_2} \mathcal{V}(G|\sigma_1, \sigma_2).$$

The *second player's value of game G with respect to* Σ_1, Σ_2 is

$$\mathcal{V}^2(G|\Sigma_1, \Sigma_2) = \inf_{\sigma_2 \in \Sigma_2} \sup_{\sigma_1 \in \Sigma_1} \mathcal{V}(G|\sigma_1, \sigma_2).$$

A game G is said to have a value with respect to strategy sets Σ_1, Σ_2 if both players' values of G are equal

$$\mathcal{V}^1(G|\Sigma_1, \Sigma_2) = \mathcal{V}^2(G|\Sigma_1, \Sigma_2).$$

This common value we then call the *value of G with respect to* Σ_1, Σ_2 and denote $\mathcal{V}(G|\Sigma_1, \Sigma_2)$. When we neglect to mention relative strategy sets the default sets are the full sets of players' strategies.

A player's strategy is optimal under pessimistic assumptions if for any reachable game state it performs no worse than any other strategy of the player in the worst-case, i.e. against the opponent's most unfavorable strategy. If they exist, we call such strategies minimax strategies. Formally, σ_1^o is a *minimax strategy of the first player with respect to* Σ_1, Σ_2 if it satisfies

$$\inf_{\sigma_2 \in \Sigma_2} \mathcal{V}(G|\sigma_1^o, \sigma_2) = \mathcal{V}^1(G|\Sigma_1, \Sigma_2).$$

Similarly, σ_2^o is a *minimax strategy of the second player with respect to* Σ_1, Σ_2 if it satisfies

$$\sup_{\sigma_1 \in \Sigma_1} \mathcal{V}(G|\sigma_1, \sigma_2^o) = \mathcal{V}^2(G|\Sigma_1, \Sigma_2).$$

A finite game, one which is guaranteed to terminate in a finite number of moves and in which the set of moves available to the players in each state is finite, is guaranteed by the famed Minimax Theorem [56] to have a value $\mathcal{V}(G) \in \Re \cup \{\infty, -\infty\}$ when mixed strategies are allowed. Both players in such a game have, possibly mixed, minimax strategies. Yet if pure strategies guarantee the game's value, then neither player can improve upon them by using mixed strategies. Such games can be played optimally under worst-case assumptions without resorting to randomization.

A computable strategy may be called an algorithm.

For an arbitrary state $S \in \mathcal{S}$ denote by $G(S)$ a game identical to G except for the starting state which is S. Let Σ_1, Σ_2 denote strategy sets of the respective players. For a strategy σ^1 let $\mathcal{S}(\sigma^1, \Sigma_2)$ denote the states $S \in \mathcal{S}$ that are reachable when the first player plays σ^1 and the second player plays a strategy in Σ_2. If for any $S \in \mathcal{S}(\sigma^1, \Sigma_2)$ strategy σ^1 guarantees the first player a payoff in game $G(S)$ that is as high as that guaranteed by any other strategy in the set Σ_1 when the second player is restricted to strategies in Σ_2 we call strategy σ^1 *opportunistic with respect to* $\langle \Sigma_1, \Sigma_2 \rangle$ *in game* G. Formally, σ^1 is opportunistic with respect to Σ_1, Σ_2 if

$$\inf_{\sigma_2 \in \Sigma_2} \mathcal{V}(G(S)|\sigma^1, \sigma_2) = \sup_{\sigma_1 \in \Sigma_1} \inf_{\sigma_2 \in \Sigma_2} \mathcal{V}(G(S)|\sigma_1, \sigma_2).$$

Similarly, a strategy of the second player $\sigma^2 \in \Sigma_2$ is opportunistic with respect to Σ_1, Σ_2 in game G if

$$\sup_{\sigma_1 \in \Sigma_1} \mathcal{V}(G(S)|\sigma_1, \sigma^2) = \sup_{\sigma_1 \in \Sigma_1} \inf_{\sigma_2 \in \Sigma_2} \mathcal{V}(G(S)|\sigma_1, \sigma_2).$$

We call a strategy *opportunistic in game* G when it is opportunistic with respect to the full sets of strategies available to the players. A strategy that is an algorithm, may be called an opportunistic algorithm.

Let $S_0, S_1, S_2, \ldots, S_k$ be the history of a game-play between the two players. Strategy σ^1 of the first player is opportunistic with respect to Σ_1, Σ_2 if for any history that is possible when the first player plays σ^1 and the second player is restricted to strategies in Σ_2 the sequence $\langle \mathcal{V}^1(G(S_i)|\Sigma_1, \Sigma_2)\rangle_{i=0}^k$ is monotonic non-decreasing. A strategy σ^2 of the second player is opportunistic if for all histories satisfying analogous conditions the sequence of values$\langle \mathcal{V}^2(G(S_i)|\Sigma_1, \Sigma_2)\rangle_{i=0}^k$ is monotonic non-decreasing.

A strategy $\sigma^1 \in \Sigma_1$ is said to be $\langle K, K' \rangle$ *almost opportunistic with respect to* $\langle \Sigma_1, \Sigma_2 \rangle$ *in game* G or K *almost opportunistic with respect to* $\langle \Sigma_1, \Sigma_2 \rangle$ *in game* G if there exist fixed real constants $K, K' \in \Re$ such that for any reachable state $S \in \mathcal{S}(\sigma^1, \Sigma_2)$,

$$\inf_{\sigma_2 \in \Sigma_2} \mathcal{V}(G(S)|\sigma^1, \sigma_2) \geq K \sup_{\sigma_1 \in \Sigma_1} \inf_{\sigma_2 \in \Sigma_2} \mathcal{V}(G(S)|\sigma_1, \sigma_2) + K'.$$

Similarly, a strategy $\sigma^2 \in \Sigma_2$ $\langle K, K' \rangle$ *almost opportunistic with respect to* $\langle \Sigma_1, \Sigma_2 \rangle$ *in game* G or K *almost opportunistic with respect to* $\langle \Sigma_1, \Sigma_2 \rangle$ *in game* G if there exist fixed real constants $K, K' \in \Re$ such that for any reachable state $S \in \mathcal{S}(\sigma^1, \Sigma_2)$,

$$\sup_{\sigma_1 \in \Sigma_1} \mathcal{V}(G(S)|\sigma_1, \sigma^2) \leq K \sup_{\sigma_1 \in \Sigma_1} \inf_{\sigma_2 \in \Sigma_2} \mathcal{V}(G(S)|\sigma_1, \sigma_2) + K'.$$

We call a strategy *almost opportunistic in* G when it is almost opportunistic with respect to the full sets of strategies available to the players. A strategy that is an algorithm, may be called an almost opportunistic algorithm.

1.3.2 Expert Consulting Games - Common Rules, Definitions

In the game we address the manager, called *Alice*, consults a set X of experts. A probability measure, π, is defined on X – that is $\pi(X) = 1$. We are interested in the worst-case analysis of Alice's algorithm and assume the experts are managed by an *adversary*. We consider those set-ups in which the advice the adversary is allowed to give through an expert at some point in the game is limited only by the count of mistakes that expert made in previous rounds. In other expert consulting set-ups limitations placed by the game's rules on the adversary's choice of an expert's advice may be limited not only by the number of mistakes each expert made, but also the rounds in which those mistakes were made. The state of the game after round i is represented by a triple

$$S_i = \langle E_i, i, e^i \rangle,$$

where $E_i \in \mathcal{N}$ is the number of errors made by Alice (zero is considered a natural number); $i \in \mathcal{N}$ is the number of *rounds* played in the game; $e^i \in [0,1]^*$ the *expert state vector*. The coordinates of e^i are real numbers that sum to one. Coordinate e^i_j is the measure of the set of experts that made j mistakes in the first i rounds.

The initial state of the game may be $S_0 = \langle 0, 0, \langle 1 \rangle \rangle$. This state represents the fact that all experts have accumulated zero errors at the start of the game expressing no preference of Alice among them. The experts that have less accumulated errors are attributed greater weight by the algorithm. Initial state vectors other than $\langle 1 \rangle$ may be used to represent a prior Alice has on the experts' quality. For "experts" who are programs or functions, the prior may be used to give a preference to simpler models of the data.

Denote by $M^i(x)$ the number of mistakes expert x is charged with after round i, and denote by X^i_j the set of experts that are charged with j mistakes in the first i rounds of the game. The number of mistakes expert x is charged with is the sum of $M^0(x)$, which corresponds to Alice's prior on x's performance, plus the number of mistakes x made in the game.

At round i, the adversary chooses vectors $n^i \in \Re^*$ representing the weight of experts voting for option one (here we assume a boolean decision domain with two options 1,2) such that $n^i \leq e^i$. The other experts, whose weight may be represented by $e - n$, are assumed to vote for the other option. We call n^i *a split*. Although game states represent only the error counts neglecting experts' identities, they may correspond to an underlying set of experts X. Thus a split n^i corresponds to a set of experts voting for option one, $X^{i,1} \subseteq X$, such that $n^i_j = \pi\{x \in X^{i,1} : M^i(x) = j\}$.

Next Alice chooses her decision $d_i \in \{1, 2\}$, after which the adversary reveals the *correct answer*. An absolute game's state vector is then updated according to the following rules:

If the correct answer is "1" then: $e^{i+1} = o^{i,1}(n^i) = n^i + (e^i - n^i) \gg 1$

If the correct answer is "2" then: $e^{i+1} = o^{i,2}(n^i) = e^i - n^i + e^i \gg 1$.

We call $o^{i,1}, o^{i,2}$ the *options* Alice is presented with.

Definition 1.3.1 *We denote by $\mathcal{G}^{XXX}$ a family of expert consulting games. The various families are distinguished by the XXX superscript. A game $G \in \mathcal{G}^{XXX}$ is specified by a tuple of up to six coordinates $\langle X, \pi, e^0, \eta, M, l \rangle$; X is the* set of experts*; π a probability measure on X; $e^0 \in \Re^*$ the expert start state satisfying $\sum e^0 = 1$; $\eta : \mathcal{N} \to [0,1]$ the share of "good" experts, those that make less than $M(i)$ mistakes in the first i rounds; $M : \mathcal{N} \to \mathcal{N}$ the Mistake Bounding function; $l \in \mathcal{N}$ is the length of the game. When* *the length is specified that means it is fixed* a-priori *independent of play. The* score *in a terminated absolute game is given by E_l. We use $G(e)$ to denote a game identical to G except, possibly, for its start state; $G(e, \pi), G(e, M)$, etc. have similar semantics. We call* Alice *(the* manager*) the player that attempts to minimize the score and her opponent* the adversary. *The notations $X[G], \pi[G], \ldots$ are used to distinguish the coordinates of the game-defining tuple.*

Notation 1.3.1 *Denote by $\mathbf{\Delta(G, V, S, l)}$ the set of adversarial strategies against which Alice is expected to make at least V mistakes in game G of length l starting at state S for any strategy she can play. Similarly, $\Lambda(\mathrm{G, V, S, l})$ is a set of Alice's strategies for which she is expected to make up to V mistakes in a game G of l steps starting at state S.*

1.3.3 Expert Consulting Games - Variations

Having discussed the commonalities of expert consulting games, we now turn to their idiosyncrasies.

Definition 1.3.2 *We call game G* non-atomic *if $\pi[G]$ is non-atomic, and* atomic *otherwise. For an atomic game we call a game with identical parameters but a non-atomic measure the* associated non-atomic game.

If we restrict our attention to sets of experts of cardinality no greater than the continuum then according to a standard theorem [58, Proposition 26.2] all non-atomic probability measures that can be defined over the set of experts $X[G]$ that agree with $\pi[G]$ are unique up to isomorphism, and isomorphic to the Lebesgue measure on the real segment $[0, 1]$. Thus, up to isomorphism, there is a single non-atomic game associated with any given game.

Definition 1.3.3 *A* game with N experts *is a game for which N is the size of the set of players, $|X| = N$; all players are assigned the same weight by the measure function $\forall x \in X \;.\; \pi(x) = \frac{1}{N}$; and function η effectively counts the number of non-erring experts $\eta : \mathcal{N} \to \{\frac{i}{N} : i = 0, 1, \cdots, N\}$.*

Families of games may vary by the knowledge the sides possess and by the amount of control they have in the game. E.g. both sides may know the length of the game in advance, just one side may know it in advance, or the power to stop the game may be given to one of the sides.

We call an absolute game *Mistake Bound (MB)* when the adversary is required to ensure that

$$e^i \cdot \mathbf{1}_{M(i)} \geq \eta(i). \tag{1.2}$$

is satisfied when the game terminates after some round i. That is, at least $\eta(i)$ of the experts make less than $M(i)$ mistakes in the first i rounds of the game. One can also explore *Prefix Mistake Bound (PMB)* games in which the adversary is required to satisfy condition (1.2) after i rounds, for all i.

Restrictions can be put on the ways in which experts' opinions interact. These can be deterministic or probabilistic. In particular splits may be selected by a stochastic process. Likewise, the correct labels may be selected by a stochastic process. To fit such games into the above framework, we can think of the adversary as being restricted to use the appropriate process to make his decisions.

One can also let $M(l)$ or $\eta(l)$ be a random variable, thus modeling a game in which these parameters are determined stochastically, although here this avenue is not explored.

The parameters of games within a family, e.g. $X, \pi, M(l), \eta(l)$ may vary from game to game.

Definition 1.3.4 *We call* game trees *the class of binary trees the nodes of which are labeled by expert state vectors, such that the children of a node labeled e are labeled by two options $o^1(n), o^2(n)$, with respect to some split $n \leq e$. For all $d \in \mathcal{N}$ the nodes at depth d satisfy the mistake bound condition (1.2) with $i = d$.*

A tree corresponds to a deterministic algorithm $\mathcal{D}$ of the adversary if the split used to label the children of a node labeled e is the split $\mathcal{D}$ chooses in state e.

1.4 Adversarial Logic

The results of this section apply to arbitrary Mistake Bound games. They also hold for Prefix Mistake Bound games in which $M(l)$ is monotonically non-decreasing, and $\eta(l)$ is monotonically non-increasing. Herein, we denote this class of games by $\mathcal{G}$. We show there are always strategies in $\Delta(G, V, S, l)$ that possess certain convenient properties. When presented with the task of making a decision about a move, Alice can assume w.l.o.g. the adversary is going to use one of these easily analyzable strategies to predict the outcome of his decision.

1.4.1 Generally Applicable Observations

Claim 1.4.1 *For any game $G \in \mathcal{G}$ and any V, S, l the set $\Delta(G, V, S, l)$ contains a pure strategy.*

The proof uses the fact that the set of maxima of a linear function on a polygon always contains a vertex of that polygon. The claim implies that restricting a computationally unlimited adversary to use pure strategies does not improve the game for Alice.

Definition 1.4.1 *We define strategy $\mathcal{A}^o$ for Alice by*

$$d^o_{\mathcal{A}^o}(o^1, o^2) = \arg\min_{i=1,2}\{v_i : v_1 = \max\{\mathcal{V}_{\mathcal{A}^o}(o^1), \mathcal{V}_{\mathcal{A}^o}(o^2)+1\}, v_2 = \max\{\mathcal{V}_{\mathcal{A}^o}(o^1)+1, \mathcal{V}_{\mathcal{A}^o}(o^2)\}\}. \tag{1.3}$$

Claim 1.4.2 *For any game $G \in \mathcal{G}$ and any V, S, l*

$$\mathcal{A}^o \in \Lambda(G, V, S, l)$$

An Alice that knows the value of any state can choose her moves optimally, by predicting her opponent's choices.

We can distinguish four types of deterministic adversarial moves in the split-choosing stage:

A-A Agreement splits. Splits for which the adversary will agree with whatever decision Alice makes.

A-D Agreement-disagreement splits. Splits for which the adversary will agree with one decision of Alice, but disagree with the other.

S Stall splits: $n(e) = e$ or $n(e) = 0^{|e|}$. The adversary agrees with the decision of the experts.

D-D Disagreement splits. For these the adversary will disagree with any decision Alice makes.

Claim 1.4.3 *For any game $G \in \mathcal{G}$, any start state S, and any game length $l \in \mathcal{N}$*

$$\mathcal{V}_{\mathcal{D}}(G(S, l)) \geq \mathcal{V}_{\mathcal{D}}(G(S, l-1)).$$

Proof: The adversary can always start the game with a stall move. □

Intuitively, vectors that are smaller with respect to the shift order can be interpreted as corresponding to more evolved positions in the expert consulting game. Thus the next claim seems to follow naturally from the previous one. It allows us to say that the value of games respects the shift order.

Claim 1.4.4 *For any game $G \in \mathcal{G}$ and two game states $S^1 = \langle E, i, a \rangle$ and $S^2 = \langle E, i, c \rangle$ such that $a \geq_S c$:*

$$\Delta(G, V, S^2, l) \neq \emptyset \;\Rightarrow\; \Delta(G, V, S^1, l) \neq \emptyset. \tag{1.4}$$

Proof: Fix a strategy $\mathcal{A}$ for Alice. Calling the number of mistakes Alice makes the adversary's payoff, we prove that

> for any pure adversarial strategy $\mathcal{D}$ there exists a pure strategy $\mathcal{D}'$ such that the expected payoff of $\mathcal{D}'$ against $\mathcal{A}$ starting with expert state vector a is at least as high as the expected payoff of $\mathcal{D}$ against $\mathcal{A}$ starting with expert state vector c.

Then Claim 1.4.1 completes the proof.

Note that only the adversary can benefit from prolonging the game. For vectors c satisfying the game termination condition the game cannot proceed and the claim holds. This establishes the base of an induction on the game's length l.

Assume the inductive statement holds for $l < L$. For $l = L$, let $\mathcal{D}_0 \in \Delta(G, V, S^2, l)$. Let n_0 be the split that $\mathcal{D}_0$ chooses in state c. Then by the definition of the shift order there exist vectors $a_1, a_2, \dots a_k$ such that

$$a \geq_S c \;\Rightarrow\; a \geq_{S1} a_1 \geq_{S1} \dots \geq_{S1} a_k \geq c.$$

By induction on k, as shown below, it follows that $\mathcal{D}'_0$ can choose a split n'_0 for state a such that both options presented to Alice by $\mathcal{D}'_0$ in state a are greater than or equal to with respect to the shift order to those presented in state c by $\mathcal{D}_0$. Hence, the inductive assumption may be applied to these options completing the proof.

For $k = 0$ let $n'_0 = n_0$.

For the inductive step, take a_1 as above. It satisfies $a \geq_{S1} a_1$, hence there exist y^1, y^2 such that

$$\begin{aligned} y^1 &+ y^2 = a \\ y^1 &+ y^2 \gg 1 = a_1. \end{aligned}$$

For a vector $b \leq a_1$ denote:

$$\begin{aligned} \hat{b}_j &= \min\{b_j, y^1_j\}, \\ b' &= \hat{b} + (b - \hat{b}) \ll 1. \end{aligned}$$

Now $\hat{b} \leq y^1$ and $b - \hat{b} \leq a_1 - y^1 = y^2 \gg 1$, hence $b' \leq a$ making b' a legal split of a. Further,

$$a - a_1 = y^2 - y^2 \gg 1 \geq_S (\hat{b} - b) - (\hat{b} - b) \gg 1 = b' - b$$

hence $a - b' \geq_S a_1 - b$ this and $b' \geq_S b$ imply together that the two options offered when b' is presented in state a, namely $a - b' + b' \gg 1$ and $(a - b') \gg 1 + b'$ are respectively bigger with respect to the shift order than the options $a_1 - b + b \gg 1$ and $(a_1 - b) \gg 1 + b$. □

It follows that

Corollary 1.4.1 *For any game $G \in \mathcal{G}$*

$$a \geq_S c \;\; \Rightarrow \;\; \mathcal{V}(G(a)) \geq \mathcal{V}(G(c)). \tag{1.5}$$

Next we show Claim 1.4.4 implies that when the adversary chooses to agree with Alice the adversary's state of affairs does not improve and may even deteriorate, provided Alice plays rationally. Thus the adversary's strategic perspectives are not harmed when he is restricted to always disagree with Alice. First observe that

Corollary 1.4.2 *For any game $G \in \mathcal{G}$ a non-empty set of Δ-strategies must contain a strategy that does not use agreement splits.*

Proof: Since stall splits are always available to the adversary, all agreement moves can be replaced by stall moves, without compromising the adversary's payoff. This follows from Claim 1.4.4, as agreement moves at least weakly decrease the state's value with respect to the shift order. □

Next we prove that A-D splits need not be used either.

Claim 1.4.5 *For any game $G \in \mathcal{G}$ and any V, S, l there is a $\Delta(G, V, S, l)$-strategy that does not use A-D splits.*

Proof: By the definition of $\mathcal{V}$:

$$\begin{aligned}\mathcal{V}(e, l|\mathcal{D}, .) = \max_{n \leq e} \min\{ \quad & \max\{\mathcal{V}(o^1, l-1|\mathcal{D}, .) + 1, \mathcal{V}(o^2, l-1|\mathcal{D}, .)\}, \\ & \max\{\mathcal{V}(o^1, l-1|\mathcal{D}, .), \mathcal{V}(o^2, l-1|\mathcal{D}, .) + 1\} \;\; \}.\end{aligned} \tag{1.6}$$

Hence, for all splits $n \leq e$

$$\min\{\mathcal{V}(o^1, l-1|\mathcal{D}, .), \mathcal{V}(o^2, l-1|\mathcal{D}, .)\} \leq \mathcal{V}(e, l|\mathcal{D}, .) - 1. \tag{1.7}$$

If a given split satisfies

$$\min\{\mathcal{V}(o^1, l-1|\mathcal{D},.), \mathcal{V}(o^2, l-1|\mathcal{D},.)\} = \mathcal{V}(e, l|\mathcal{D},.) - 1$$

then from the definition of the game's value w.l.o.g. the adversary can use it as a D-D split. This is preferable to using it as an A-D split. Thus a rationally chosen A-D split must satisfy

$$\min\{\mathcal{V}(o^1, l-1|\mathcal{D},.), \mathcal{V}(o^2, l-1|\mathcal{D},.)\} \le \mathcal{V}(e, l|\mathcal{D},.) - 2.$$

In fact by the definition of the game's value (1.6) it must satisfy

$$\begin{aligned}
\min\{\mathcal{V}(o^1, l-1|\mathcal{D},.), \mathcal{V}(o^2, l-1|\mathcal{D},.)\} &= \mathcal{V}(e, l|\mathcal{D},.) - 2, \\
\max\{\mathcal{V}(o^1, l-1|\mathcal{D},.), \mathcal{V}(o^2, l-1|\mathcal{D},.)\} &= \mathcal{V}(e, l|\mathcal{D},.).
\end{aligned}$$

The second equation follows from the first by the definition of the game's value. Equality holds by Claims 1.4.3 and 1.4.4. Assume w.l.o.g.

$$\mathcal{V}(o^1, l-1|\mathcal{D},.) = \mathcal{V}(e, l|\mathcal{D},.).$$

This means that Alice can decide 1 rationally leading to an agreement. By Claim 1.4.4 if the adversary presents the stall split $n' = e$

$$\begin{aligned}
&\max\{\mathcal{V}(o^1, l-1|\mathcal{D},.), \mathcal{V}(o^2, l-1|\mathcal{D},.) + 1\} \\
&\quad = \max\{\mathcal{V}(e, l-1|\mathcal{D},.), \mathcal{V}(e \gg 1, l-1|\mathcal{D},.) + 1\} \\
&\quad = \mathcal{V}(e, l|\mathcal{D},.). \\
&\max\{\mathcal{V}(o^1, l-1|\mathcal{D},.) + 1, \mathcal{V}(o^2, l-1|\mathcal{D},.)\} \\
&\quad = \max\{\mathcal{V}(e, l-1|\mathcal{D},.) + 1, \mathcal{V}(e \gg 1, l-1|\mathcal{D},.)\} \\
&\quad = \mathcal{V}(e, l|\mathcal{D},.) + 1.
\end{aligned}$$

Alice still decides 1 rationally, leaving the game in state e. By Claim 1.4.4 $e \ge_S o^1$ implies this move is no worse for the adversary. □

Thus we have proven:

Theorem 1.4.1 *For any game $G \in \mathcal{G}$ and any V, S, l there is a $\Delta(G, V, S, l)$-strategy that performs only S and D-D moves.*

The adversary can determine the rounds of the game at which he executes the D-D moves. In a Mistake Bound Game, there is no reason to delay their execution. Thus a *generic optimal adversarial strategy* for MB games is to execute rationally justified D-D moves while available, and then stall to the end of the game. In particular if no D-D moves will become available in the future, the adversary and Alice can choose to stop the game without incurring a loss.

In a general Prefix Mistake Bound game the optimal strategy of the adversary might need to take into account global considerations when deciding whether to execute a D-D or S move. Yet if $M(l)$ is non-decreasing and $\eta(l)$ non-increasing D-D moves can be executed whenever they become available, as a legal game state cannot become illegal when the game advances. Alternatively, for arbitrary $M(l), \eta(l)$ the adversary can execute all his D-D moves consecutively at the last $\mathcal{V}(G)$ moves of the game. This can be proven by induction using the fact that a D-D move that is followed by an S move can be swapped with it without violating any mistake bound restrictions.

Looking at game trees, we have so far argued that we only need to consider trees with D-D and S moves to compute the value of a game. The value of a game is given by the $\max_{\text{trees}} \min_{\text{leaves}}$ of the number of D-D moves on the path to the leaf.

1.4.2 Non-Atomic Games

Pascal Matrices

Definition 1.4.2 *Denote by P_l^n a* Pascal Matrix *of size $n \times n$, order l. The entry in row i and column j, for $0 \le i < n$ and $0 \le j < n$ of this matrix is*

$$[P_l^n]_{i,j} = \binom{l}{i-j}.$$

Comment: For $c < 0$ and $c > l$ we have $\begin{pmatrix} l \\ c \end{pmatrix} = 0$.

We index rows and columns of matrices starting at 0. We may neglect to mention the size of the matrix, n, as it is often not essential in our application, provided n is sufficiently large (larger than M).

Here, for example, are a couple 4×4 Pascal Matrices:

$$P_2^4 = \frac{1}{4} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 1 & 2 & 1 & 0 \\ 0 & 1 & 2 & 1 \end{pmatrix}, \quad P_0^4 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

For a vector a and any matrix Q

$$Q\,(a \gg 1) = (Q \ll 1)\,a.$$

Since for an $n \times n$ matrix Q with $Q = [q_{ij}]$ and $a = \langle a_1, \ldots, a_n \rangle$

$$[q_{ij}] \cdot (\langle a_1, \ldots, a_n \rangle \gg 1) = \langle \sum_{j=2}^{n} q_{ij} a_{j-1} \rangle_{i=1}^{n} = (Q \ll 1)\,a.$$

Now observe that

$$\frac{1}{2} P_l + \frac{1}{2} P_l \ll 1 = P_{l+1}. \tag{1.8}$$

It follows from the facts quoted above that:

$$\begin{aligned} P_l\,(\frac{1}{2}a + \frac{1}{2}a \gg 1) &= P_l\,\frac{1}{2}a + P_l\,(\frac{1}{2}a \gg 1) \\ &= \frac{1}{2}P_l\,a + \frac{1}{2}(P_l \ll 1)\,a \\ &= (\frac{1}{2}P_l + \frac{1}{2}P_l \ll 1)\,a \\ &= P_{l+1}\,a. \end{aligned}$$

Thus

$$P_l\,(\frac{1}{2}a + \frac{1}{2}a \gg 1) = P_{l+1}\,a. \tag{1.9}$$

An Optimal Adversarial Strategy for Non-Atomic Games

We call an adversary's move a *"half" move* if it presents the D-D split $\frac{1}{2}e$ at state e.

Claim 1.4.6 *If the adversary executes l "half" moves beginning with state e then the game moves into state $P_l e$, regardless of Alice's strategy.*

Proof: By induction on l using equation (1.9). □

Notation 1.4.1 *Denote by $\mathcal{D}_{\frac{1}{2}}$ the adversarial strategy that executes a "half" move in state e if that does not cause a violation of the mistake bound condition, and that stalls otherwise.*

Theorem 1.4.2 *For non-atomic game $G \in \mathcal{G}$ if $\Delta(G, V, S, l) \neq \emptyset$ then $\mathcal{D}_{\frac{1}{2}} \in \Delta(G, V, S, l)$.*

Proof: For a deterministic strategy $\mathcal{D}$ of the adversary that executes D-D and stall moves only, let us look at the tree representing $\mathcal{D}$'s executions. We have argued that $\mathcal{V}(G|\mathcal{D}, .) \geq V$ is the minimal number of D-D moves between a depth l node and the root. Replacing D-D moves by stall moves on branches Alice does not take rationally does not cause a violation of the mistake bound condition (1.2), thus we can assume w.l.o.g. that on any path from the root to a depth l leaf there are V D-D moves. Now remove all nodes corresponding to stall moves from the tree. Assuming Alice votes with all of her consultants in such nodes, and that the adversary announces her and her consultants right we can attach the subtree rooted at the child that is labeled identically to the parent-node in place of the parent-node. Denote by $e_1^V, \ldots, e_{2^V}^V$ the nodes at depth V. By induction on V it can be shown that:

$$\sum e_1^l = 2^V P_V e_0$$

where e_0 is the start state, as for any two options at a state e

$$o^1 + o^2 = e + e \gg 1.$$

For all $i = 1, \ldots, 2^V$:

$$e_i^V \cdot \mathbf{1}_M \geq \eta.$$

Thus

$$\sum_i (e_i^V \cdot \mathbf{1}_M) = (2^V P_V e_0) \cdot \mathbf{1}_M \leq 2^V \eta.$$

By Claim 1.4.6 had the adversary played $\mathcal{D}_{\frac{1}{2}}$ he would have executed at least V successive "half" moves during the first V rounds of the game. The resulting state would have satisfied the mistake bound condition at termination securing him a payoff of V. □

1.4.3 Games with Finitely Many Experts

In non-atomic games the adversary has more splits to choose from than in the corresponding atomic games. Thus

Theorem 1.4.3 *For two games $G_1, G_2 \in \mathcal{G}$ identical, except for the measure function, such that $\pi[G_1]$ is atomic and $\pi[G_2]$ is non-atomic*

$$\mathcal{V}_{\mathcal{D}}(G_1) \leq \mathcal{V}_{\mathcal{D}}(G_2).$$

Next we restrict our attention to games with finitely many experts. The adversary is not much worse off in these games than in the associated non-atomic games.

Consulting Finitely Many Experts in Mistake Bound Games

Notation 1.4.2 *Denote by* n$^{\mathrm{a}}$ *the split defined by lining up the experts in order of decreasing number of accumulated errors (ties are broken arbitrarily) and choosing into n^a those experts at odd positions. Call it the* alternating split.

These splits were considered by Spencer and Winkler [67].

Corollary 1.5.1 of Section 1.5.1 states that the value of a non-atomic game with constant mistake bound M and adversely determined length is given by (the same notation is used in both places):

$$V^{MB,cM}(G(e)) = \arg\max_l \{P_l\, e \cdot \mathbf{1}_M \geq \eta\}.$$

in the remainder of this section we denote $V(e, M, \eta) = V^{MB, cM}(G(e))$.

Theorem 1.4.4 *For two Mistake Bound games G_1, G_2 identical, except for the measure function, such that G_1 is a game with finitely many experts and G_2 is non-atomic*

$$\mathcal{V}_{\mathcal{D}}(G_1) \geq \lceil \frac{1}{2} \mathcal{V}_{\mathcal{D}}(G_2) \rceil.$$

Proof: By Theorems 1.4.1 and 1.4.2 in the non-atomic game G_2 w.l.o.g. the adversary executes

$$V = V(e, M, \eta) = \arg\max_{l} \{P_l \, e \cdot \mathbf{1}_M \geq \eta\} = \mathcal{V}_{\mathcal{D}}(G_2(e))$$

"half" moves, for some M, and then stalls until the game ends. Denote the number of errors the least erring expert made getting to state e by

$$i_{LE}(e) = \arg\min_{i} \{e_i \neq 0\}.$$

The argument now splits into two cases :

Case 1 : [$2(M - i_{LE}(e) - 1) \geq V$] — In the first case the least erring expert is allowed to make $M - i_{LE}(e) - 1$ additional errors. The adversary can use this to cause the manager to incur at least this many mistakes, without violating the mistake bound requirement. Thus for state vectors satisfying $2(M - i_{LE}(e) - 1) \geq V$ the theorem holds.

Case 2 : [$2(M - i_{LE}(e) - 1) \leq V - 1$] — In Case 2 we want to prove the theorem by induction on V. For vectors covered by case 1 the base is established. Yet some states with $V = 1$ might not satisfy $2(M - i_{LE}(e) - 1) \geq V$, i.e. $i_{LE}(e) = M - 1$ that is $\sum e = e_{M-1}$. Since $V = 1$ the adversary can perform another "half" move, thus

$$\sum e - \frac{1}{2} e_{M-1} \geq \eta.$$

Since e is a state in a game with finitely many experts, $e_{M-1}, \eta \in \{\frac{i}{N} : i \in \mathcal{N}\}$, implying

$$\sum e - \lceil \frac{1}{2} e_{M-1} \rceil \geq \eta.$$

Hence the adversary can play another D-D move in a game with finitely many experts as well.

Now let us complete the inductive proof of case 2.

Induction's step : Having established the base we prove for the step that for any e, M, N:

$$V(e, M, \eta) - 2 \leq \min\{V(o^1(n^a), M, \eta),\ V(o^2(n^a), M, \eta)\}.$$

By the definition of V

$$\frac{1}{2^V}\langle \binom{V}{< (M-i)} \rangle_{i=0}^{M-1} \cdot e \geq \eta \geq \frac{1}{N}. \tag{1.10}$$

Denoting

$$e_{LE} = \langle \frac{1}{N} \rangle \gg i_{LE}$$

the bound $2(M - i_{LE} - 1) \leq V - 1$ implies that no more than half the terms in the sequence $\binom{V}{0}, \binom{V}{1}, \ldots, \binom{V}{V}$ are summed up on the left-hand side of the following expression giving

$$\langle \binom{V}{< (M-i)} \rangle_{i=0}^{M-1} \cdot e_{LE} \leq \frac{1}{2N}.$$

Thus by (1.10):

$$\langle \binom{V}{< (M-i)} \rangle_{i=0}^{M-1} \cdot e_{LE} \leq \langle \binom{V}{< (M-i)} \rangle_{i=0}^{M-1} \cdot (e - e_{LE}). \tag{1.11}$$

The ratio

$$\binom{V}{c} \Big/ \binom{V-1}{c} = \frac{V}{V-c}$$

increases as c grows. Thus

$$\langle \frac{\binom{V}{< (M-i)}}{\binom{V-1}{< (M-i)}} \rangle_{i=0}^{M-1} \cdot e_{LE} \ \geq \ \langle \frac{\binom{V}{< (M-i)}}{\binom{V-1}{< (M-i)}} \rangle_{i=0}^{M-1} \cdot (e - e_{LE})$$

We get from (1.11):

$$\langle \binom{V-1}{<(M-i)} \rangle_{i=0}^{M-1} \cdot e_{LE} \leq \langle \binom{V-1}{<(M-i)} \rangle_{i=0}^{M-1} \cdot (e - e_{LE}),$$

which is equivalent to

$$\sum_{j=1}^{M-i_{LE}} \binom{V-1}{M-i_{LE}-j} \leq \sum_{j=1}^{M-i_{LE}} \langle \binom{V-1}{M-i-j} \rangle_{i=0}^{M-1} \cdot (e - e_{LE}).$$

The ratio

$$\binom{V}{c} / \binom{V}{c-1} = \frac{V-c+1}{c}$$

decreases as c grows. Hence,

$$\binom{V-1}{M-i_{LE}-1} \leq \langle \binom{V-1}{(M-i-1)} \rangle_{i=0}^{M-1} \cdot (e - e_{LE}). \tag{1.12}$$

Now note that for any game state e

$$\begin{aligned}
\sum (P_{l-1} - P_l)e &= \langle \frac{1}{2^{l-1}} \binom{l-1}{<(M-i)} - \frac{1}{2^l} \binom{l}{<(M-i)} \rangle_{i=0}^{M-1} \cdot e \\
&= \frac{1}{2^l} \langle \binom{l-1}{<(M-i)} - (\binom{l}{<(M-i)} - \binom{l-1}{<(M-i)}) \rangle_{i=0}^{M-1} \cdot e \\
&= \frac{1}{2^l} \langle \binom{l-1}{<(M-i)} - \binom{l-1}{<(M-i-1)} \rangle_{i=0}^{M-1} \cdot e \\
&= \frac{1}{2^l} \langle \binom{l-1}{M-i-1} \rangle_{i=0}^{M-1} \cdot e
\end{aligned}$$

Hence

$$\begin{aligned}
\sum P_{V-2}(o^1(n^a) + o^2(n^a)) &= \sum P_{V-2}(e + e \gg 1) \\
\text{(by (1.9))} &= \sum 2P_{V-1}e \\
&= \sum [2P_V e + 2(P_{V-1} - P_V)e]
\end{aligned}$$

$$\begin{aligned}
\text{(by the definition of } V) &\geq 2\eta + \frac{1}{2^{V-1}} \langle \binom{V-1}{M-i-1} \rangle_{i=0}^{M-1} \cdot e \\
\text{(by (1.12)} &\geq 2\eta + \frac{1}{2^{V-2}} \binom{V-1}{M-i_{LE}-1}. \qquad (1.13)
\end{aligned}$$

On the other hand

$$|\sum P_{V-2}(o^1(n^a) - o^2(n^a))| = \frac{1}{2^{V-2}} \langle \binom{V-2}{M-i-1} \rangle_{i=0}^{M-1} \cdot (2n^a - e). \qquad (1.14)$$

The condition $2(M - i_{LE} - 1) \leq V - 1$ implies

$$\max_i \binom{V-2}{M-i-1} = \max\{\binom{V-2}{M-i_{LE}-1}, \binom{V-2}{M-i_{LE}-2}\}.$$

Hence, for the sum of a sequence with alternating signs

$$\begin{aligned}
|\sum P_{V-2}(o^1(n^a) - o^2(n^a))| &\leq \frac{1}{2^{V-2}} \max\{\binom{V-2}{M-i_{LE}-1}, \binom{V-2}{M-i_{LE}-2}\} \\
&\leq \frac{1}{2^{V-2}} \binom{V-1}{M-i_{LE}-1}.
\end{aligned}$$

Together with (1.13) this means:

$$\min\{\sum P_{V-2}o^1(n^a), \sum P_{V-2}o^2(n^a)\} \geq \eta$$

completing the proof of the inductive step. □

Consulting Finitely Many Experts in Prefix Mistake Bound Games

Let us denote by $E^{\frac{1}{2}\cdot k}(e)$ the set of states that can result from the adversary executing k "half" moves starting at state e. By Claim 1.4.6 $|E^{\frac{1}{2}\cdot k}(e)| = 1$. Similarly, $E^{a\cdot k}(e)$ is the set of all possible outcome states for k steps of the alternating strategy. It follows from the proof of Theorem 1.4.4 that

Corollary 1.4.3 *For a state vector e in a game with N experts, and for arbitrary $k, M \geq 0$ and $i > 0$ let $e^a \in E^{a \cdot \lceil k/2 \rceil}(e), e^{\frac{1}{2}} \in E^{\frac{1}{2} \cdot k}(e)$ then*

$$e^{\frac{1}{2}} \cdot \mathbf{1}_M \geq \frac{i}{N} \Rightarrow e^a \cdot \mathbf{1}_M \geq \frac{i}{N}.$$

Proof: By applying Theorem 1.4.4 to the N experts' game with $\eta = \frac{i}{N}$ and the associated non-atomic game. □

This implies that

Theorem 1.4.5 *For two Prefix Mistake Bound games G_1, G_2, identical except for the measure function, such that G_1 is a game with finitely many experts and G_2 is non-atomic*

$$\mathcal{V}_{\mathcal{D}}(G_1) \geq \lceil \frac{1}{2} \mathcal{V}_{\mathcal{D}}(G_2) \rceil.$$

Proof: Let $V = \mathcal{V}_{\mathcal{D}}(G_2(e))$. By Theorem 1.4.2 we can assume w.l.o.g. that in the non-atomic game, G_2, the adversary executes only stall moves and V "half" moves. Referring to the "half" moves in this sequence as first, second, etc., the adversary can replace the D-D "half" splits at odd positions by D-D alternating splits. By Corollary 1.4.3 this is a legitimate adversarial strategy achieving a payoff of $\lceil \frac{1}{2} V \rceil$. □

1.5 Games' Values and Managerial Strategies

In the previous section it is shown that in an expert consulting game the adversary may be restricted to a small set of strategies without changing his worst-case payoff. This section looks at the conclusions a manager can derive from the understanding of her adversary that we gained.

1.5.1 The Values of Games

Theorem 1.4.2 and Claim 1.4.6 give a simultaneous upper an lower bound on the game's value and allow an easy computation of the value of various families of games. We give a few examples:

Theorem 1.5.1 *The value of a non-atomic MB game G, of length l is given by*

$$V^{MB}(G,l) = \arg\max_{k\le l}\{P_k\, e^0 \cdot \mathbf{1}_{M(l)} \ge \eta(l)\}. \tag{1.15}$$

Theorem 1.5.2 *The value of a non-atomic MB game G, of bounded length $\le l$ that is determined by the adversary is given by*

$$V^{MB,bl}(G,l) = \arg\max_{k}\max_{\{i:k\le i\le l\}}\{P_k\, e^0 \cdot \mathbf{1}_{M(i)} \ge \eta(i)\}. \tag{1.16}$$

Notice that for a constant mistake bound $M(l) = M, \eta(l) = \eta$ the adversary's payoff is upper bounded by an expression that remains bounded as $l \to \infty$. Thus

Corollary 1.5.1 *A non-atomic MB game G, with a constant mistake bound, in which the adversary gets to determine the length of the game has a value of*

$$V^{MB,cM}(G) = \arg\max_{l}\{P_l\, e^0 \cdot \mathbf{1}_M \ge \eta\}. \tag{1.17}$$

1.5.2 The Manager's Strategy

Knowing the value of game states gives Alice the power to play the game opportunistically, as proven for the managerial strategy introduced in Definition 1.4.1. Having given explicit computable formulas of the values of games in the previous section, we can use these to present computationally efficient algorithms for expert managers. We term the algorithms presented in this section the *PM algorithm*, after the matrices they resort to.

Non-Atomic Games

Notation 1.5.1 *Denote by $\mathcal{A}^{MB}, \mathcal{A}^{MB,bl}, \mathcal{A}^{MB,cM}$ the strategies defined by*

$$d^o_{\mathcal{A}^{XXX}}(o^1,o^2) = \arg\min_{i=1,2}\{v_i = \max\{V^{XXX}(o^i), V^{XXX}(o^{3-i})+1\}\}. \tag{1.18}$$

The XXX superscript can be replaced appropriately.

Theorem 1.5.3 *Strategies $\mathcal{A}^{MB}, \mathcal{A}^{MB,bl}, \mathcal{A}^{MB,cM}$ are opportunistic in the respective non-atomic games.*

Proof: By Claim 1.4.2 and Theorems 1.5.1, 1.5.2 and Corollary 1.5.1. $\square$

Games with Finitely Many Experts

Theorem 1.5.4 *Strategies $\mathcal{A}^{MB}, \mathcal{A}^{MB,bl}, \mathcal{A}^{MB,cM}$ are $\langle 2, 0\rangle$ almost opportunistic in the respective games with finitely many experts.*

Proof: Expert consulting games are zero sum, thus $\mathcal{V}_{\mathcal{D}} = \mathcal{V}_{\mathcal{A}} = \mathcal{V}$. Therefore, the theorem follows from Theorems 1.4.3 and 1.4.4. $\square$

Strategies $\mathcal{A}^{XXX}$ reach their decision by comparing the values of the respective V^{XXX} functions. They can be refined to compare in state e the values of $\sum P_{V^{XXX}(e)-1} \cdot o^1$ and $\sum P_{V^{XXX}(e)-1} \cdot o^2$. These refined algorithms reach the same decision as the original algorithms whenever $V^{XXX}(o^1) \neq V^{XXX}(o^2)$. However, they allow a better comparison of options which are equivalent in the associated non-atomic game. Yet even the refined algorithms do not always reach the optimal decision, as demonstrated by the following example. Thus they are not V-optimal for games with finitely many experts. The problem of finding such algorithms that are efficient remains open. Section 1.7.4 discusses the close relationship between expert consulting games and the better investigated faulty search games. In the latter opportunistic algorithms for finitely many candidate values are known only for the cases $M = 1$ (see Pelc [59]) and $M = 2$ (see Guzicki [30]).

Example (Even the refined algorithms are not opportunistic): Consider a Constant Mistake Bound game with six experts, and $\eta = \frac{1}{6}$.

$$\mathcal{V}(\frac{1}{6}\langle 2, 3, 0, 0, 1\rangle) = 9,$$

$$V^{MB,cM}(\frac{1}{6}\langle 2, 3, 0, 0, 1\rangle) = 10.$$

Now if the adversary presents the split

$$n = \frac{1}{6}\langle 0, 3, 0, 0, 1\rangle,$$

then the value of the two options, based on the findings of a computer program, is

$$\mathcal{V}(\frac{1}{6}\langle 2,0,3,0,0\rangle) = 9, \quad \mathcal{V}(\frac{1}{6}\langle 0,5,0,0,1\rangle) = 8.$$

Yet, as can be readily verified

$$V^{MB,cM}(\frac{1}{6}\langle 2,0,3,0,0\rangle) = V^{MB,cM}(\frac{1}{6}\langle 0,5,0,0,1\rangle) = 9.$$

If Alice uses the refined method to reach her decision, she finds that

$$\sum P_9 \cdot \frac{1}{6}\langle 0,5,0,0,1\rangle = \frac{651}{6\cdot 2^9}, \quad \sum P_9 \cdot \frac{1}{6}\langle 2,0,3,0,0\rangle = \frac{650}{6\cdot 2^9},$$

leading her to the wrong decision.

1.6 Efficiency Issues

Next we discuss the efficient implementation of the PM algorithms and their asymptotic performance. We also refer to the lower bounds implied by their opportunism.

1.6.1 From Strategies to Algorithms

Alice can use the theorems in section 1.5.2 as explained in section 1.5.1. When the adversary presents Alice with a set of options Alice needs to compute and compare the values of the appropriate V^{XXX} function for these options to reach a decision.

The value of the initial state vector e_0 can be found by repeated doubling – comparing $P_l e$ to η for a sequence of matrices with $l = 2^0, 2^1, 2^2, \ldots$, and then once $P_l e$ becomes smaller than η for the first time performing a binary search for the value of l for which $P_l e < \eta$ holds for the first time between the last two values of l for which $P_l e$ was evaluated. The search thus takes at most $2 \lg V^{XXX}(e_0)$ steps. Each step requires computing M matrix values and multiplication by a vector. The factorial terms in each matrix entry require $O(M)$ multiplications each, for a total of $O(M^2)$ computational steps per matrix on numbers of size exponential in M (multiplication can be carried out in poly-logarithmic time). The bounds on the value of states in section 1.6.2 are polynomial in M and $\lg \frac{1}{\eta}$. Thus the off-line precomputation can be carried out in polynomial time.

After the off line pre-computation of $V^{XXX}(e_0)$ was carried out, the values of P_{l-1} can be computed from those in P_l in $O(M)$ multiplications. At state e the value of least one of the two options presented to Alice is at least $V^{XXX}(e) - 1$ by (1.7). Therefore, Alice needs to multiply the options by one matrix only. At most two matrix-vector multiplications are thus required to reach a decision. If the adversary plays suboptimally, additional computations might be needed to compute the pseudo value of the new state.

The complexity of an on-line step can be reduced further by converting PM to an equivalent weighting scheme. Note that coordinate n_i of the split "votes" for both options. It votes for option one with weight $\begin{pmatrix} l \\ \leq i \end{pmatrix} n_i$ and for option two with weight $\begin{pmatrix} l \\ \leq (i-1) \end{pmatrix} n_i$. Thus letting the experts of n_i vote with weight $\begin{pmatrix} l \\ i \end{pmatrix} n_i$ for decision 1 and letting the experts in $e_i - n_i$ vote with weight $\begin{pmatrix} l \\ i \end{pmatrix}(e_i - n_i)$ for decision 2, summing up these weights for all coordinates of n and voting with the heavy set leads Alice to the same decision.

Corollary 1.6.1 *The PM strategies are computationally efficient.*

1.6.2 Absolute Performance

Opportunism reflects the performance of algorithms relative to other algorithms for the problem. Herein we obtain absolute loss bounds. Proving these algorithms are opportunistic means that these bounds are also lower bounds on the performance of any algorithm for the problem. They hold even for computationally unlimited algorithms and for the expected performance of randomized algorithms.

Theorem 1.6.1 *The total loss of algorithm $\mathcal{A}_{MB}^{PM}$ in a non-atomic MB game of length l with a uniform prior on experts is*

$$\arg\max_{k \leq l} \{ \begin{pmatrix} k \\ M(l) \end{pmatrix} \geq \eta(l) \}. \tag{1.19}$$

Any other strategy's expected loss for this problem is at least as high on some inputs.

Theorem 1.6.2 *The total loss of algorithm $\mathcal{A}^{PM}_{MB,cM}$ in a non-atomic MB game with $M(l) = M$ and a uniform prior is*

$$\arg\max_{l}\{l \leq \lg\frac{1}{\eta} + \lg\begin{pmatrix} l \\ \leq M \end{pmatrix}\} = \lg\frac{1}{\eta} + M\lg\lg\frac{1}{\eta} + O(M\lg M). \qquad (1.20)$$

Any other strategy's expected loss for this problem is at least as high on some inputs.

(The right side of (1.20) is due to Rivest et al. [61].)

1.7 Extensions and Implications

The results proven thus far for expert consulting games can be generalized in various ways. They have interesting parallels to those obtained for the problem of searching in the presence of errors. New proofs of old results as well as some novel results in that area follow from what we have shown.

1.7.1 Consulting Finitely Many Experts: Comparing PM to BW and WM

The PM algorithms are more general than the previously known WM and BW algorithms. In particular its performance guarantees hold for arbitrary initial expert state vectors. This can be used to incorporate a prior the manager has on his/her consultants. Such a prior can represent an initial estimate of the quality of the experts' predictions. It can also be based on other parameters like experts' salaries, or in the case of model selection the candidate models' complexity. Our algorithms further allow $\eta \neq \frac{1}{N}$, while previously known algorithms are restricted to the case of tracking the best expert.

If we limit our discussions to games of tracking the best expert with a uniform prior, then PM is $\langle 2, 0\rangle$ almost opportunistic, while the previously known algorithms are not. Littlestone and Warmuth [44] suggest WM can be used in schemes that update the weights of experts in every round, or only in rounds in which the manager errs for the same asymptotic performance. An important difference between BW and PM is that unlike PM, BW updates the weights only in rounds in which the algorithm errs. The superiority of PM provides evidence in favor of weight updates in every round.

The upper bounds on the worst-case asymptotic performance of PM are the same as those of BW. Cesa-Bianchi et al. [18] argue they are superior to those of WM. The next example proves that the opportunistic ratio of BW is unbounded.

Example (BW's opportunistic ratio is unbounded): Let us define a family $\{G_k\}$ of expert consulting games. Game G_k is a game with $n = 2^k + 1$ experts in which the best expert is known not to err.

Consider the following adversarial strategy: In the first k rounds the adversary splits the experts that did not err yet into two almost equal subsets. The difference between their sizes is exactly one. The experts that erred in previous rounds join the bigger set. Both BW and PM vote with the bigger set of experts, which the adversary announces correct in these rounds. Thus by the end of round k, the manager and a single "good" expert incurred no errors while all other experts erred once.

The value of the subgame starting after round k is zero as the identity of the non-erring expert was already revealed. The PM algorithm "knows" which expert is "good", while BW ignores the information contained in the opening rounds.

In the next k rounds the experts split into two subsets the size of which differs by one, the "good" expert voting with the minority. BW makes k mistakes in these rounds, while PM votes correctly. Thus while we have proven PM is $\langle 2, 0 \rangle$ almost opportunistic BW is not such, as the ratio between the number of mistakes made by BW and the number of mistakes made by an opportunistic algorithm can be arbitrarily large.

1.7.2 Consulting Experts on Multiple Choice Questions

So far we considered decision domains of size 2. That is the advice of the experts, as well as Alice's decision, answered a "yes/no" question. In this section we prove that the PM algorithms as defined in section 1.5.2 are opportunistic or almost opportunistic when the advice of the experts as well as Alice's decision come from a finite set D of arbitrary size $|D| \geq 2$.

For domains of size ≥ 2 a split of vector e would be a $|D|$-tuple of vectors in $\Re^*$, $n = \langle n^1, n^2, \ldots, n^D \rangle$, such that $\sum n^i = e$. The respective options are now defined to be $o^i = n^i + (e - n^i) \gg 1$.

The adversary can play strategy $\mathcal{D}_{\frac{1}{2}}$ by choosing splits $n = \langle \frac{e}{2}, \frac{e}{2}, 0^*, \ldots, 0^* \rangle$ in the game with an arbitrary decision domain D. Thus the value of the game is lower bounded by its pseudo value $\mathcal{V}(G(e)) \geq V^{XXX}(e)$. To prove the opposite inequality we notice that the proofs of the Claims of section 1.4 generalize to games with bigger decision domains. Thus the properties of the pseudo value functions established in section 1.5.1 still hold, allowing Alice to use the PM algorithms in games with bigger decision domains for the same performance guarantees.

1.7.3 "Real" Managers

Alice may be allowed to make decisions in the domain $[0, 1]$. If she decides d while the correct answer turns out to be c she is charged a loss of $|d - c|$, known as the absolute loss. If Alice's strategy is to make random binary predictions with probability d of predicting one, then the absolute loss measures Alice's expected loss in the game.

Notice that if we replace Alice's $[0, 1]$ decisions by binary decisions in the obvious way, making her vote one whenever $d \geq \frac{1}{2}$, then the loss she incurs is at most twice the loss she would incur if she were allowed to make decisions in $[0, 1]$. Thus the strategies described in section 1.5.2 are $\langle 2, 0 \rangle$ almost opportunistic in a non-atomic game and $\langle 4, 0 \rangle$ almost opportunistic in an atomic game.

1.7.4 Searching in the Presence of Errors

The problem of searching interactively under the assumption that some answers may be erroneous is well investigated. It was first introduced by Ulam [69] and addressed by numerous researchers [61, 60, 22, 30, 24, 6, 66, 67, 5]. Most of these papers model it by a multistage game. The searcher is commonly called *Paul* and his adversary *Carole*. The problem most of these papers address is searching for a single value in a finite domain. In Rivest et al. [61] and Pelc [60] the authors consider finding the ϵ vicinity of a single real value in a given segment.

The various versions of games of searching by questioning a liar using arbitrary membership queries are comparable to expert consulting games. The representation of the state in such games is the same as in expert-consulting games, but the protocol differs. In a search game the searcher presents a query to Carole in the form of a subset of the set of candidate values. Carole replies whether the target value is in this subset or its complement. Thereby she decides the values in which one of the two subsets will accumulate one more vote against them being the target value. Some authors associate each candidate value with a *chip*, hence the name *"chip games"* [6]. The chips are positioned in a sequence of piles on a ray. The allowed positions for the various piles on this ray are indexed by the natural numbers. Paul is called a *chooser*, for his role of choosing the questioned subset, while Carole a *pusher* as she selects the subset of chips which will be pushed one position ahead on this ray representing the values associated with these chips having incurred an extra vote against them being the target value. The game proceeds in rounds. Various limitations are placed on the ways in which Carole is allowed to lie. She might be allowed a constant number of lies [61, 59]. Games in which Carole is allowed to lie $\lfloor rl \rfloor$ times in a game of l rounds are addressed by Aslam and Dhagat [6] and Spencer and Winkler [67]. This limitation might be enforced when the game terminates, or at each round.

In the expert consulting game the adversary (Carole) chooses a subset of X. Alice decides which set she want to vote with, and then the adversary announces Alice right or wrong. Unlike Paul, Alice is restricted to choose the queried set from the two candidates presented by her adversary. On the other hand, while Paul is charged for each question he poses, Alice's "questions" can be interpreted as "guesses". Accordingly, she gets charged for bad guesses only.

Suppose we restrict the adversary in a Mistake Bound to always disagree with Alice's decision, and give him the power to stop the game at any legal state he chooses. By Theorem 1.4.1 this does not change the game's value. Now the adversary becomes the chooser (Paul) in a chip game and Alice the pusher (Carole). However, the chooser's aim in this game is to prolong the game, while the pusher tries for the opposite. This is a reversal of goals as compared to the goals of Paul and Carole. In a faulty search game the chooser is the one trying to shorten the game.

Recall the class of game trees from Definition 1.3.4. The value of an expert game is

$$\max_{\text{trees}} \min_{\text{leaves}} \text{depth(leaf)},$$

while the value of a Paul-Carole game is

$$\min_{\text{trees}} \max_{\text{leaves}} \text{depth(leaf)}.$$

Alice, thus has to make no more mistakes than Paul. For non-atomic games the values of the two games are equal, as "Paul" can ask about "half" splits.

Our approach generalizes that common in papers addressing the faulty search problem in that it allows the searcher to express a non-uniform prior on the set of candidate values, by choosing arbitrary start vectors, not only $\langle 1 \rangle$. We further analyze larger classes of $M(l)$ and $\eta(l)$ functions. We thus extend results of Rivest et al. [61] for what they call continuous games.

Mistake bounded adversaries making linearly many mistakes, $M(l) = \lfloor rl \rfloor$ against membership queries of Paul were explored by Spencer and Winkler [67]. We show

Theorem 1.7.1 *Paul needs at least $\frac{4 \ln N}{(1-2r)^2}$ questions to find the hidden number out of N candidates in a Mistake Bound (Version A of Spencer and Winkler [67]) game.*

Proof: A Faulty Search Game with finitely many candidate values is related to a non-atomic search game, in a relationship similar to that existing in Expert Consulting Games. The value of the associated non-atomic game is a lower bound (rather than an upper bound as in expert consulting games) on the value of a game with finitely many candidate values. The value of a non-atomic Faulty Search Game is in turn is lower bounded by the value of the associated Expert Consulting Game as explained above. Thus the number of questions

Paul needs to pose to find one out of N candidate values is at least

$$\arg\max_{l}\{P_l\langle 1\rangle \cdot 1_{\lfloor rl\rfloor} \geq \frac{1}{N}\} = \arg\max_{l}\{2^{-l}\binom{l}{<\lfloor rl\rfloor} \geq \frac{1}{N}\}. \tag{1.21}$$

Now

$$2^{-l}\binom{l}{<\lfloor rl\rfloor} = \Pr\{< \lfloor rl\rfloor \text{ successes in } l \text{ throws of an unbiased coin}\}.$$

Using Chernoff's bound [38]:

$$\Pr[S_l < (1-\gamma)pl] \leq e^{-lp\gamma^2/2}.$$

we let $p = 1/2$, $\gamma = 1-2r$ and get

$$e^{-l(1-2r)^2/4} \geq e^{-\ln N}$$

or

$$l \leq \frac{4\ln N}{(1-2r)^2}.$$

□

Spencer [66] implicitly proposes to evaluate states in the search game by equation (1.17) for $\eta = 1$. His rationale for using these weights stems from considering a randomized strategy of Carole in which she uses a fair coin to decide her answers. The probability of a chip to advance no more than s positions in j rounds is then $\binom{j}{\leq s}2^{-j}$. Equation (1.17) arises by asking whether the expected number of chips on the game board after a given number of steps is at least one.

1.7.5 The Relative Game

In this section we consider a game in which Alice's score is the number of mistakes she made in excess of her best advisors rather than the absolute number of mistakes she made. The states' representation remains unchanged but the semantics of states in the relative game is different. The j-th coordinate of the expert state vector now represents the measure of the set of experts that made j mistakes less than Alice. Thus when split n^i is presented at some game state e^i the game can move into one of four states. If Alice's decision is one the game can move into one of the states:

$$e^{i+1} = n^i + (e^i - n^i) \gg 1,$$

$$e^{i+1} = n^i + (e^i - n^i) \ll 1.$$

Two analogous states are possible if she makes the other decision.

If the adversary is required to satisfy the game termination condition, the game becomes a win/lose game in which the adversary either does or does not have a strategy that meets the requirements.

A game termination condition may still be imposed as in absolute games. Alternatively, the adversary may not be required to satisfy a game termination condition in the relative game. The score in a relative game can then be defined

$$\arg\max_M \{e^i \cdot \mathbf{1}_M \leq \eta(i)\}. \tag{1.22}$$

That is Alice is scored by the number of mistakes she made in excess of a prespecified share η of her advisors that make the smallest number of mistakes of all of her advisors in the game. Her aim is to minimize the score.

The analysis of section 1.4 still holds, by similar argumentation. We can assume w.l.o.g. that the adversary always disagrees with Alice's decision, showing the problem of consulting experts in the relative game is reducible to the faulty search problem (a Paul-Carole search game).

The equivalent of Theorem 1.4.2 likewise holds for relative games, stating that strategy $\mathcal{D}_{\frac{1}{2}}$ is opportunistic in non-atomic games. Thus the value of non-atomic relative games can be computed efficiently, as in Theorems 1.5.1, 1.5.2 and Corollary 1.5.1. E.g.

Theorem 1.7.2 *The value of a relative game of known length l with an unspecified termi-*

nation condition is given by

$$\arg\max_{M} P_l e^0 \cdot \mathbf{1}_M \leq \eta(l).$$

This in turn allows to specify computationally efficient opportunistic managerial strategies in the non-atomic game.

If a winning strategy for the adversary (Paul) exists in a game with finitely many experts, such a strategy clearly exists in the associated non-atomic game, giving a necessary condition for the victory of the adversary. Corollary 1.4.3 establishes a sufficient condition giving us

Theorem 1.7.3 *In a Mistake Bound game with N experts if*

$$P_l e^0 \cdot \mathbf{1}_M > \eta(l) \tag{1.23}$$

then Alice wins. If

$$P_{2l} e^0 \cdot \mathbf{1}_M \leq \eta(l)$$

then the adversary wins.

This establishes the same necessary condition for Paul's (the adversary's) victory as that in Spencer's [66] and a different sufficient condition.

Notice that

$$\binom{l}{< M} \leq c l^M.$$

Thus for l, c sufficiently large depending on M if $e^0_{M-1} > c l^M$ Condition 1.23 becomes sufficient as well as necessary, as the adversary can use the most erring experts counted by coordinate e^0_{M-1} (those of weight 1 in the vector $P_l e^0$) to choose splits for which the weight of both associated options is equal throughout the game. He can split alternatingly all but the most erring players, and use those to balance the weight of the two options. Thereby he will effectively play a strategy equivalent to $\mathcal{D}_{\frac{1}{2}}$. This gives a different proof of the validity of the sufficient condition (the main result) specified by Spencer [66].

Based on computer experiments the author conjectures that when presented with two options o^1, o^2 in a relative game with finitely many experts the manager (pusher) can make an optimal decision by comparing the value of the same states in the associated non-atomic game, except for state vectors for which the least erring expert made significantly fewer mistakes than the other experts. The evaluation function for non-atomic games attaches such experts excessive weight. Instead the value of such states in a game with finitely many experts is equal to the value of an almost identical state in which the least erring expert is charged a single additional error. Proving this conjecture will reduce the problem of computing the exact value of states in the relative game with finitely many experts, addressed by Pelc [59] and Guzicki [30] for $M = 1$ and $M = 2$ respectively to the problem of calculating the number of moves in a game with prespecified strategies for both sides.

1.8 Conclusion

Chip games were explored previously as a model of a faulty search procedure. We show they can be used to model expert-consulting situations as well. A chip game in which the goals of the pusher and chooser are exchanged is investigated as a model of another variant on the expert consulting problem. Our exploration of these games proceeds via an analysis of what we propose to call non-atomic chip games. Both games with finitely many experts which correspond to those chip games described in the existing literature, and non-atomic games are instances of a more general model of interest – games in which an arbitrary measure is defined over the set of chips.

We derive the exact value of non-atomic chip games. For previously explored chip-games, corresponding to our games with finitely many experts, a similar result is known only for games with $M = 1, 2$, while the general question remains unanswered. The specification of opportunistic strategies, or better yet algorithms, for both players in these games is a problem we leave open. Computationally it is no less interesting than the older problem of figuring out the value of such games, and can serve as a stepping stone towards that problem's resolution.

Acknowledgments

Ron Rivest suggested to the author the problem of consulting experts and the manager's name – Alice.

Chapter 2

Analysis of Greedy Expert Hiring and an Application to Memory-Based Learning

2.1 Introduction and Definitions

One of the great challenges in modeling human capacity is overcoming redundant information. Some researchers conjecture this is the main function of the early processing carried out by the brain (Marr [47]). Extracting that which is essential is likewise a difficult algorithmic problem arising in various circumstances - the clique problem, traveling salesman and many more. In the context of computational learning it was addressed explicitly by Blum [16], Blum et al. [15], Littlestone [43], Ben-David and Dichterman [10, 11], Birkendorf et al. [14].

Consider a manager faced with the task of hiring experts from a pool of N candidates. We assume that he can find out the utility of hiring particular sets of experts by querying an oracle $\mathbf{x} : \mathbf{2}^N \to \Re$. Finding an optimal solution of size k would force him to look at $\begin{pmatrix} n \\ k \end{pmatrix}$ possible sets of experts. This number is exponential in k. Therefore, looking for an optimal solution in the general case is infeasible for large k.

The manager may choose to substitute global considerations for local ones by hiring, for example, one expert at a time greedily. That is, hiring at step j the expert that contributes most to the set of $j-1$ experts that were already hired. This would reduce the complexity (number of calls to the oracle) to a reasonable $\Theta(kn)$.

The optimism of the greedy heuristic can be contrasted with the pessimism underlying worst-case analysis. The author believes that most conscious data processing is carried out by simple heuristics. Thus a detailed understanding of the conditions governing the performance of such heuristics will contribute no less to our understanding of conscious intelligence than for example the exploration of complex optimization algorithms.

For an unrestricted input no performance guarantees can be provided for this heuristic. However, for functions $\mathbf{x}$ that are monotone and concave a uniform lower bound on the performance of greedy hiring holds. Nemhauser et al. [52] present a bound on the ratio between the value of the greedy approximation and that of the optimal solution.

When the value of sets of experts has to be estimated by sampling, the manager may only have access to approximate values of such sets, rather than exact values. We show a uniform lower bound on the quality of a greedy approximation for the same family of games is still valid.

This can model, for example, a learning situation in which the learner has access to some "learning engine" as a subroutine and to a source of labeled examples. The learner (manager) can draw a labeled sample, and then run the learning subroutine on a filtering of that sample that passes to the learning subroutine only the values of the coordinates in some chosen subset of coordinates. He can find out or estimate the "value" of the various subsets of coordinates by testing the hypothesis this subroutine produces. His goal is to choose a good subset of the coordinates (select features) while minimizing the number of invocations of the learning subroutine.

The analysis of the performance of greedy hiring in coalitional games implies a lower bound on approximations of the s-median problem defined below. Approximation algorithms for the s-median problem are in turn, is a useful tool in the development of a learning algorithm for Lipschitz functions (Lin and Vitter [42]).

A *memory-based learning system* is a system that approximates (learns) a given function, $f : X \to Z$, in the following manner: An instance of the input space X is mapped by an *encoder* γ to the addresses of one or more memory locations. The contents of these locations are combined by a *decoder* β to produce an output in Z. Learning can be done in batch mode or on-line. The encoder or the decoder or both of them can be learned. A memory-based learning system can be evaluated by its sample, time, and space complexities.

Conceivably, such systems can be used to learn functions over both discrete and continuous domains. Lin and Vitter [42] give a historical overview of early research on memory-based learning systems. Their stated main result is a memory-based learning system that PAC-learns in polynomial time and space, to which we propose an alternative.

A *Voronoi system* is a very simple memory-based learning system. The system can be specified by s pairs $\{\langle x_i, z_i \rangle\}_{i=1}^{s}$, where $x_i \in X, z_i \in Z$. The encoder maps a point x to the index of its nearest neighbour in $\{x_i\}_{i=1}^{s}$, say i_0 if the nearest point is x_{i_0}. The decoder outputs z_{i_0}. The x_i-s do not have to be stored explicitly. We call s the *size* of the system.

A function f satisfies the *Lipschitz condition* if there exists a constant K such that

$$(\forall x, x' \in X) \ . \ d_Y(f(x), f(x')) \leq K d_X(x, x'),$$

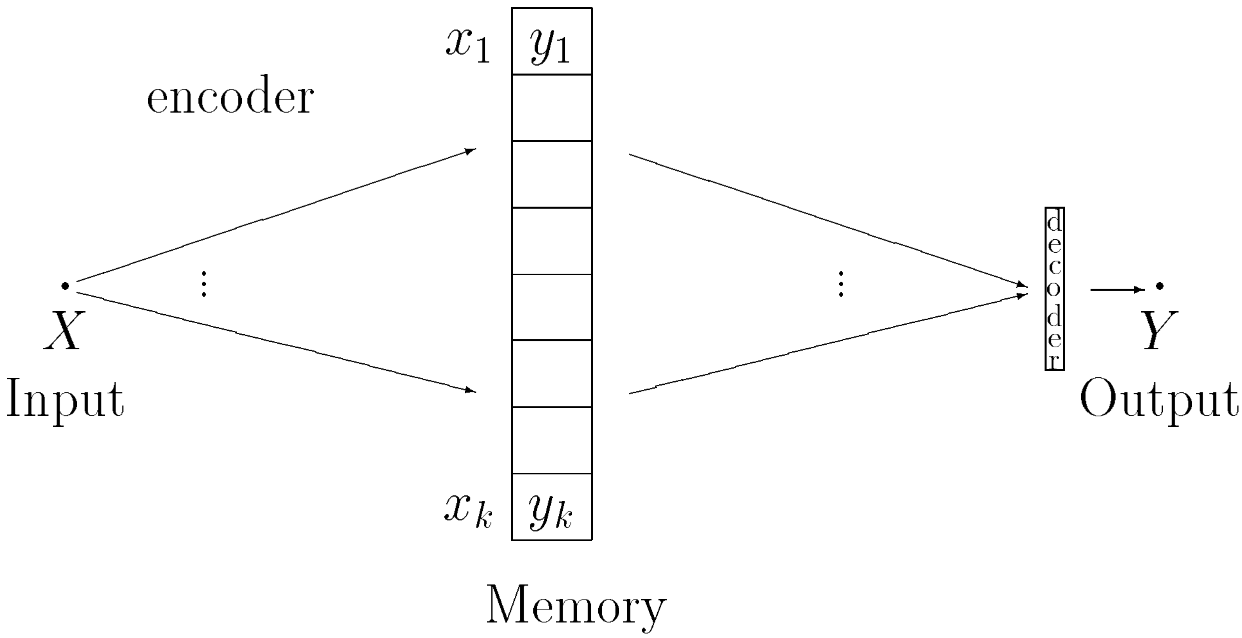

Figure 2-1: A memory-based learning system.

where d_X, d_Y are metrics defined over spaces X and Y respectively. A function satisfying this condition is called a *Lipschitz function* with bound K. A class of functions $\mathcal{F}$ is *uniformly Lipschitz bounded* if there exists a bound K such that all functions in $\mathcal{F}$ are K-Lipschitz functions. Call such an $\mathcal{F}$ a *class of Lipschitz functions.*

Now let us define *the s-median problem* (based on Lin and Vitter [41]). In section 2.4 we discuss its relevance to efficient memory-based learning of Lipschitz functions. The input is a complete (directed or undirected) graph $G = (V, E)$ on n vertices. Non-negative weights c_{ij} are associated with the edges. We call the c_{ij}-s *distances.* The goal is to choose a subset U of size s of the vertices that minimizes the sum of distances from each vertex to its nearest neighbour in U. Call U the *median set.*

The s-median problem arises in data compression, network location, and clustering. It is $\mathcal{NP}$-hard even in the Euclidian space [48, 57]. Lund and Yanakakis's lower bounds for the set-covering problem imply that it is $\mathcal{NP}$-hard to find ϵ-approximate solutions of size $o(s \log |V|)$ to the s-median problem for an ϵ sufficiently small [41, 46].

We generalize the analysis of Cornuejols et al. [21] to account for approximate solutions of the s-median problem that are not necessarily of size s. We then evaluate the usage of a greedy approximation scheme as an alternative to the (also greedy) approximation scheme used by Lin and Vitter [42] in their algorithm for memory-based learning of Lipschitz functions. It is found to be easier to implement and to have better time complexity than the scheme proposed by Lin and Vitter. In many cases it also yields a smaller Voronoi system.

Section 2.2 describes known and novel results stating conditions for successful greedy expert hiring. Section 2.3 reviews the Lin-Vitter approximation algorithm for the s-median problem, and presents and analyzes a simpler greedy alternative. Section 2.4 describes how these algorithms can be used as part of a memory-based learning system, and compares them in that context.

2.2 Greedy Expert Hiring

This section establishes conditions that guarantee lower bounds on the performance of the greedy heuristic when it is applied to expert hiring problems. It uses mathematical tools from the theory of coalitional games which it first recounts. Subsection 2.2.2 describes results proven by operations researchers that are relevant to hiring when the exact values of the various sets of experts are known. The following subsection extends these bounds to the situation when these values are known only approximately.

2.2.1 Coalitional Games, Concave Coalitional Games

We begin by reviewing a few definitions and facts about coalitional games. A coalitional game is a function:

$$v : \mathbf{2}^N \rightarrow \Re$$

satisfying $v(\emptyset) = 0$. The set $N = \{1, \ldots, n\}$ is commonly called the *set of players*, and the set of its subsets, $\mathbf{2}^N$, the set of *coalitions*. (Here we assume N is finite.) An introductory text on coalitional game theory is by Owen [56].

A game is *monotone* if $\forall S, T \subseteq N$ such that $S \subseteq T$:

$$v(S) \leq v(T).$$

A game is *additive* if $\forall S, T \subseteq N, S \cap T = \emptyset$:

$$v(S) + v(T) = v(S \cup T).$$

A game is *subadditive* if $\forall S, T \subseteq N, S \cap T = \emptyset$:

$$v(S) + v(T) \geq v(S \cup T).$$

A game is *concave* if it satisfies the *condition of diminishing returns* for all $i \in N$ and for all S, T such that $S \subseteq T \subseteq N \setminus \{i\}$:

$$v(S \cup \{i\}) - v(S) \geq v(T \cup \{i\}) - v(T).$$

These naming conventions are due to Shapley [63] who defined and investigated convex games. After Shapley we justify the name "concave games" by defining a differencing operator Δ_R for all $R, S \subseteq N$:

$$[\Delta_R v](S) = v(S \cup R) - v(S \setminus R).$$

If we let $\Delta_{QR} v$ denote $\Delta_Q(\Delta_R v)$ then the definition of concavity given above is equivalent to the assertion that these "second differences" are everywhere negative, i.e. $\forall Q, R, S \subseteq N$

$$[\Delta_{QR} v](S) \leq 0$$

The operator Δ_{QR} is analogous to the second derivative associated with concave functions in real analysis.

An equivalent definition of concave games is via the condition

$$\forall S, T \subseteq N \;.\; v(S) + v(T) \geq v(S \cup T) + v(S \cap T).$$

It follows that all concave games are subadditive.

Two games, v, u defined on the same set of players N are termed *equivalent* if their difference is an additive game, that is, if there exist constants $v_1, \ldots, v_n$ such that $\forall S \subseteq N$

$$v(S) - u(S) = \sum_{i \in S} v_i.$$

Any game equivalent to a concave game is concave. In particular a scalar multiple of a concave game is concave. Hence the set of concave games for a fixed N forms a convex cone in the linear space $\Re^{\mathbf{2}^N - \{\emptyset\}}$. This cone contains the subspace of additive games.

2.2.2 Hiring Experts Using Exact Values of Coalitions

Let N denote the set of experts. For a subset S of the experts, where $S \subseteq N$, let $\mathbf{x}(S)$ represent the value of this coalition for the manager. In the following we call $\mathbf{x}$ the *coalitional expert game*.

Assume the manager has access to an oracle $\mathbf{x}$ that he can query for the value $\mathbf{x}(S)$ of an arbitrary coalition S. If the manager is interested in hiring a set of k consultants it is natural for him to try a greedy approach. This means that he repeatedly hires the locally optimal expert. Starting with an empty set of hired experts, the first expert to be hired is the expert e_1 maximizing $\mathbf{x}(\{e_1\})$. After j experts $e_1^{gr}, \ldots, e_j^{gr}$ have been selected the next expert, e_{j+1}^{gr}, to be hired is the one satisfying

$$e_{j+1}^{gr} \in \arg\max\{\mathbf{x}(e_1^{gr}, \ldots, e_j^{gr}, \bar{e}) : \bar{e} \in N \setminus \{e_1^{gr}, \ldots, e_j^{gr}\}\}.$$

We would like to derive a bound on the performance of this greedy heuristic that is uniformly valid for a family of games. That is find a bound valid for all games in the family. Let $\{e_1^{opt}, \ldots, e_k^{opt}\}$ be an optimal set of k experts, and let X_k^{opt} be its value:

$X_k^{opt} = \mathbf{x}(e_1^{opt}, \ldots, e_k^{opt})$. Let $\{e_1^{gr}, \ldots, e_j^{gr}\}$ be a set of j greedily hired experts of value $X_j^{gr} = \mathbf{x}(e_1^{gr}, \ldots, e_j^{gr})$. We would like the ratio

$$P_{j,k} = X_j^{gr}/X_k^{opt}$$

to be lower bounded by a function of j and k that does not depend on $\mathbf{x}$.

This problem was considered by Nemhauser and Wolsey [51, 50] and by Nemhauser, Wolsey and Fisher [52]. They prove that for games that are monotone and concave

$$P_{k,k} \geq 1 - (1 - \frac{1}{k})^k > 1 - \frac{1}{e}.$$

A similar bound can be proven on $P_{j,k}$ for arbitrary j. They also analyze a somewhat more sophisticated and general approximation scheme for which they prove a matching inverse bound:

Theorem 2.2.1 *For a concave and monotone coalitional game, and for each integer $q > 0$ there is an algorithm that uses $O(n^{q+1})$ queries of oracle $\mathbf{x}$ and finds a coalition E^q of arbitrary size k for which*

$$P_k^q = \frac{\mathbf{x}(E^q)}{X_k^{opt}} \geq 1 - (1 - \frac{q}{k})(1 - \frac{q}{k-1})^{k-q}.$$

For any integer $q > 0$, P_k^q is the best ratio achievable by an algorithm that uses $O(n^{q+1})$ queries to find a coalition of size k.

2.2.3 Hiring Experts Using Approximate Values of Coalitions

In practice the assumption that the manager has access to an oracle providing him with the precise values of expert coalitions might be unrealistic. In some situations the values of coalitions may have to be estimated by a stochastic process, e.g. by experiments the manager performs with those coalitions. The manager then might have to make his decisions based on approximate values of coalitions, rather than on exact values. This section establishes a uniform lower bound on the performance of a greedy manager in these circumstances. We model the situation by letting the manager query an oracle that gives approximations of the coalition's values.

An *α-approximate oracle* for game $\mathbf{x}$, denoted $\mathbf{x}^{\sim\alpha}$, is an oracle that when queried with a coalition S of players returns arbitrary values satisfying

$$\alpha\mathbf{x}(S) \leq \mathbf{x}^{\sim\alpha}(S) \leq \alpha^{-1}\mathbf{x}(S) \tag{2.1}$$

for $0 < \alpha \leq 1$.

We prove that

Theorem 2.2.2 *For a coalition of size j, $E_j^{gr,\sim\alpha}$, that was hired greedily with respect to an α-approximate coalitional game oracle $\mathbf{x}^{\sim\alpha}$ of a concave and monotone game $\mathbf{x}$*

$$\begin{aligned} \mathbf{x}(E_j^{gr,\sim\alpha}) &\geq \max_{k=1,\ldots,n}\{(1-(1-\frac{\alpha^2}{k})^j)X_k^{opt}\} \\ &\geq \max_{k=1,\ldots,n}\{(1-e^{-\alpha^2 j/k})X_k^{opt}\}. \end{aligned}$$

The proof of this theorem is presented at the end of the section.

Definition 2.2.1 *For $0 < \alpha \leq 1$ call a coalition $\{e_1^{gr,\alpha},\ldots,e_j^{gr,\alpha}\}$* α-greedily hirable *if it can be ordered $\hat{e}_1^{gr,\alpha},\ldots,\hat{e}_j^{gr,\alpha}$ so that for all l, $0 \geq l < j$:*

$$\mathbf{x}(\hat{e}_1^{gr,\alpha},\ldots,\hat{e}_l^{gr,\alpha},\hat{e}_{l+1}^{gr,\alpha}) \geq \alpha \max_{\bar{e}} \mathbf{x}(\hat{e}_1^{gr,\alpha},\ldots,\hat{e}_l^{gr,\alpha},\bar{e}). \tag{2.2}$$

Claim 2.2.1 *A coalition $\{e_1^{gr,\alpha},\ldots,e_j^{gr,\alpha}\}$ is α-greedily hirable with respect to game oracle $\mathbf{x}$ if and only if it greedily hirable with respect to a $\sqrt{\alpha}$-approximate oracle for the game $\mathbf{x}$.*

Proof: $\Longrightarrow$ Let $e_1^{gr,\alpha},\ldots,e_j^{gr,\alpha}$ be the ordering with respect to which coalition

$$E_j^{gr,\alpha} = \{e_1^{gr,\alpha},\ldots,e_j^{gr,\alpha}\}$$

is α-greedily hirable. Define a $\sqrt{\alpha}$-approximate oracle $\mathbf{y}$ for the game $\mathbf{x}$. It returns $\alpha^{-\frac{1}{2}}\mathbf{x}(S)$ for the j coalitions of the form $S_m = \{e_1^{gr,\alpha},\ldots,e_m^{gr,\alpha}\}$ where $m \leq j$. It returns $\alpha^{\frac{1}{2}}\mathbf{x}(S)$

for all other coalitions. Then from equation (2.2) it follows that coalition $E_j^{gr,\alpha}$ is greedily hirable with respect to oracle $\mathbf{y}$.

$\Longleftarrow$ If for two coalitions we have $\mathbf{x}^{\sim\beta}(S) \leq \mathbf{x}^{\sim\beta}(T)$ then

$$\beta\mathbf{x}(S) \leq \mathbf{x}^{\sim\beta}(S) \leq \mathbf{x}^{\sim\beta}(T) \leq \beta^{-1}\mathbf{x}(T)$$

implies

$$\mathbf{x}(T) \geq \beta^2\mathbf{x}(S).$$

Thus greedy hiring with respect to $\mathbf{x}^{\sim\alpha^{\frac{1}{2}}}$ yields α-greedy hiring with respect to $\mathbf{x}$. $\square$

Denote by $\mathcal{E}_j^{gr,\alpha}$ the set of all α-greedily hirable coalitions and let

$$X_j^{gr,\alpha} = \min_{E\in\mathcal{E}_j^{gr,\alpha}} \mathbf{x}(E),$$

$$P_{j,k}^{\alpha} = X_j^{gr,\alpha}/X_k^{opt}.$$

Theorem 2.2.3 *For expert games which are concave and monotone and any $0 < \alpha \leq 1$ and integers j, k*

$$P_{j,k}^{\alpha} \geq 1 - (1 - \frac{\alpha}{k})^j > 1 - e^{-\alpha j/k}.$$

Neither concavity alone nor monotonicity alone guarantee a non-trivial bound on $P_{j,k}^{\alpha}$ that is uniformly valid for all games.

Proof: Let $E_j^{gr,\alpha} = \{e_1^{gr,\alpha}, \ldots, e_j^{gr,\alpha}\}$ denote one of the worst α-greedily hirable coalition of j experts. Assume w.l.o.g. $e_1^{gr,\alpha}, \ldots, e_j^{gr,\alpha}$ are ordered to satisfy (2.2). For $l = 0, \ldots, j-1$

$$\begin{aligned}
& X_k^{opt} - \mathbf{x}(e_1^{gr,\alpha}, \ldots, e_l^{gr,\alpha}) \\
&= \mathbf{x}(\{e_1^{opt}, \ldots, e_k^{opt}\}) - \mathbf{x}(\{e_1^{gr,\alpha}, \ldots, e_l^{gr,\alpha}\}) \\
(\mathbf{x} \text{ is monotone}) \quad &\leq \mathbf{x}(\{e_1^{gr,\alpha}, \ldots, e_l^{gr,\alpha}, e_1^{opt}, \ldots, e_k^{opt}\}) \\
& \quad -\mathbf{x}(\{e_1^{gr,\alpha}, \ldots, e_l^{gr,\alpha}\}) \\
&= [\mathbf{x}(\{e_1^{gr,\alpha}, \ldots, e_l^{gr,\alpha}, e_1^{opt}, \ldots, e_{k-1}^{opt}, e_k^{opt}\}) \\
& \quad -\mathbf{x}(\{e_1^{gr,\alpha}, \ldots, e_l^{gr,\alpha}, e_1^{opt}, \ldots, e_{k-1}^{opt}\})]
\end{aligned}$$

$$
\begin{aligned}
& +[\mathbf{x}(\{e_1^{gr,\alpha},\ldots,e_l^{gr,\alpha},e_1^{opt},\ldots,e_{k-2}^{opt},e_{k-1}^{opt}\}) \\
& \quad -\mathbf{x}(\{e_1^{g,\alpha},\ldots,e_l^{gr,\alpha},e_1^{opt},\ldots,e_{k-2}^{opt}\})] \\
& \ldots \\
& +[\mathbf{x}(\{e_1^{gr,\alpha},\ldots,e_l^{gr,\alpha},e_1^{opt}\}) - \mathbf{x}(\{e_1^{gr,\alpha},\ldots,e_l^{gr,\alpha}\})] \\
(\mathbf{x} \text{ is concave}) \quad \leq \quad & [\mathbf{x}(\{e_1^{gr,\alpha},\ldots,e_l^{gr,\alpha},e_k^{opt}\}) - \mathbf{x}(\{e_1^{gr,\alpha},\ldots,e_l^{gr,\alpha}\})] \\
& +[\mathbf{x}(\{e_1^{gr,\alpha},\ldots,e_l^{gr,\alpha},e_{k-1}^{opt}\}) - \mathbf{x}(\{e_1^{gr,\alpha},\ldots,e_l^{gr,\alpha}\})] \\
& \ldots \\
& +[\mathbf{x}(\{e_1^{gr,\alpha},\ldots,e_l^{gr,\alpha},e_1^{opt}\}) - \mathbf{x}(\{e_1^{gr,\alpha},\ldots,e_l^{gr,\alpha}\})] \\
\leq \quad & \frac{k}{\alpha}[\mathbf{x}(\{e_1^{gr,\alpha},\ldots,e_l^{gr,\alpha},e_{l+1}^{gr,\alpha}\}) - \mathbf{x}(\{e_1^{gr,\alpha},\ldots,e_l^{gr,\alpha}\})]
\end{aligned}
$$

We get that

$$\mathbf{x}(e_1^{gr,\alpha},\ldots,e_{l+1}^{gr,\alpha}) \geq (1-\frac{\alpha}{k})\mathbf{x}(e_1^{gr,\alpha},\ldots,e_l^{gr,\alpha}) + \frac{\alpha}{k}X_k^{opt},$$

or

$$\frac{\mathbf{x}(e_1^{gr,\alpha},\ldots,e_{l+1}^{gr,\alpha})}{X_k^{opt}} \geq (1-\frac{\alpha}{k})\frac{\mathbf{x}(e_1^{gr,\alpha},\ldots,e_l^{gr,\alpha})}{X_k^{opt}} + \frac{\alpha}{k}.$$

By induction on l it follows that:

$$\frac{\mathbf{x}(E_j^{gr,\alpha})}{X_k^{opt}} = P_{j,k}^{\alpha} \geq 1-(1-\frac{\alpha}{k})^j > 1-e^{-\alpha j/k}.$$

We now show that even for $\alpha = 1$ neither condition can be dropped.

Concavity is necessary : For an arbitrary natural number M consider the following game $\mathbf{x}_0$ that is monotone but not concave:

$$
\begin{aligned}
& \mathbf{x}_0(\emptyset) = 0, \\
& \mathbf{x}_0(\{1\}) = 1, \;\; \mathbf{x}_0(\{2\}) = 0, \;\; \mathbf{x}_0(\{3\}) = 0, \\
& \mathbf{x}_0(\{1,2\}) = 1, \;\; \mathbf{x}_0(\{2,3\}) = M, \;\; \mathbf{x}_0(\{1,3\}) = 1, \\
& \mathbf{x}_0(\{1,2,3\}) = M.
\end{aligned}
$$

Then $X_2^{gr} = 1$, $X_2^{opt} = M$, giving the ratio $P_{2,2} = P_{2,2}^1 = \frac{1}{M}$ that is not bounded from below.

Monotonicity is necessary : For an arbitrary natural M consider game $\mathbf{x}_1$ that is concave but not monotone:

$$\begin{aligned}
&\mathbf{x}_1(\emptyset) = 0,\\
&\mathbf{x}_1(\{1\}) = M+1,\ \ \mathbf{x}_1(\{2\}) = M,\ \ \mathbf{x}_1(\{3\}) = M,\\
&\mathbf{x}_1(\{1,2\}) = 1,\ \ \mathbf{x}_1(\{2,3\}) = 2M-1,\ \ \mathbf{x}_1(\{1,3\}) = 1,\\
&\mathbf{x}_1(\{1,2,3\}) = M-1.
\end{aligned}$$

Then $X_2^{gr} = 1$, $X_2^{opt} = 2M-1$. Hence, $P_{2,2} = P_{2,2}^1 = \frac{1}{2M-1}$ can be arbitrarily small. $\square$

The bound on $P_{j,k}^{\alpha}$ holds simultaneously for optimal solutions of all sizes. This allows us to state a stronger lower bound on the performance of α-greedy hiring.

Corollary 2.2.1 *For an α-greedily hirable coalition $E_j^{gr,\alpha}$ of size j in a concave and monotone game $\mathbf{x}$:*

$$\begin{aligned}
\mathbf{x}(E_j^{gr,\alpha}) &\geq \max_{k=1,\dots,n}\{(1-(1-\frac{\alpha}{k})^j)X_k^{opt}\}\\
&\geq \max_{k=1,\dots,n}\{(1-e^{-\alpha j/k})X_k^{opt}\}.
\end{aligned}$$

We conclude this section by proving Theorem 2.2.2.

Proof (of Theorem 2.2.2): The theorem follows from Claim 2.2.1 and Corollary 2.2.1. $\square$

2.3 Two Approximation Algorithms for the s-Median Problem

Lin and Vitter [42] present an algorithm that finds an approximate solution to the s-median problem by solving a linear programming problem and then applying the greedy heuristic to the solution. Using the results in the previous section we analyze the performance of the greedy heuristic when applied to the same problem directly and then compare the bounds these two approaches yield.

2.3.1 The Lin-Vitter Algorithm, Review

For a point x in a metric space X and a set $S \subseteq X$ it is common to define the distance of x to S as

$$d_X(x, S) = \inf[d_X(x, y) : y \in S].$$

The *s-median problem* is the problem given a finite set of points ξ, of finding a subset of ξ of size s called the *median set* for which the average distance of points in ξ to the median set is minimum.

The s-median problem for a set ξ of m points $\xi = \{x_1, \ldots, x_m\}$ can be formulated as a $0-1$ integer program of minimizing

$$\hat{d}_\xi(U) = \frac{1}{m}\sum_{i=1}^{m} d_X(x_i, U) = \frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{m} p_{ij} d_X(x_i, x_j)$$

subject to

$$\begin{aligned} \textstyle\sum_{j=1}^{m} p_{ij} &= 1, \quad && i = 1, \ldots, m; \\ \textstyle\sum_{j=1}^{m} q_j &\leq s; && \\ p_{ij} &\leq q_j, && i, j = 1, \ldots, m; \\ p_{ij}, q_j &\in \{0, 1\}, && i, j = 1, \ldots, m; \end{aligned}$$

where $q_j = 1$ iff x_j is chosen as a *cluster center*, and $p_{ij} = 1$ iff $q_j = 1$ and x_i is "assigned" to x_j's *cluster*.

The linear program relaxation of the above is allowing q_j and p_{ij} to take arbitrary values in the interval $[0, 1]$. The value of an optimal fractional solution (linear program solution) is a lower bound on the value of solutions of the discrete s-median problem.

The Lin-Vitter algorithm works as follows:

1. Solve the linear program relaxation of the discrete s-median problem by linear programming techniques; denote the fractional solution by $\hat{q}, \hat{p}$.

2. For each $i = 1, \ldots, m$ compute $\hat{D}_i = \sum_{j=1}^{m} d_X(x_i, x_j)\hat{p}_{ij}$.

3. Given a relative error bound $\epsilon > 0$, for each j such that $\hat{q}_j > 0$, construct a set S_j: A point x_i is in S_j iff $d_X(x_i, x_j) \leq (1+\epsilon)\hat{D}_i$. (Note that $x_j \in S_j$ for all S_j.)

4. Apply the greedy set cover algorithm [19, 36] to the covering of ξ by the sets $\{S_j\}$, choosing iteratively the set S_j that covers the most uncovered points. Repeat this process until all points of ξ are covered. Let I_U be the set of indices of sets chosen by the greedy set-covering heuristic. Output $U = \{x_i\}_{i \in I_U}$ as the median set.

The linear programming problem can be solved in provably polynomial time by the ellipsoid algorithm [39] or by the interior point method [37]. The simplex method [23] works very efficiently in practice, although in the worst case its performance is not polynomial-time.

Lin and Vitter [41] show:

Theorem 2.3.1 *Given any $\epsilon > 0$, the Lin-Vitter algorithm outputs a set U of size at most*

$$s(1+1/\epsilon)(\ln m + 1)$$

such that

$$\hat{d}_\xi(U) \leq (1+\epsilon)\hat{D},$$

where $\hat{D}$ is the average distance of the optimal fractional solution for the discrete s-median problem.

2.3.2 A Simple and Efficient Greedy Algorithm

Cornuejols et al. [21] were the first to derive a bound on the performance of the greedy heuristic for the s-median problem. The bound they showed is somewhat stronger than the bound in Theorem 2.3.2, as explained towards the end of this subsection. Subsequently, Nemhauser et al. [52] generalized the result to arbitrary coalitional games. We show a derivation of the bound for the s-median problem from the general bound for Coalitional Games. We also generalize their analysis to allow approximation of the s-median set by sets of size other than s.

Description of the Algorithm.

Given a set $\xi = \{x_i\}_{i=1}^{m}$ of m points in X and a nonnegative integer t, where $t \le m$, the algorithm selects a subset of size t of ξ. The algorithm works as follows:

1. Set $S = \emptyset$

2. Choose $\bar{x} \in \arg\min_{x \in \xi \setminus S} \hat{d}_{\xi}(S \cup \{x\})$ and set $S = S \cup \{\bar{x}\}$.

3. Repeat step 2 t times.

The time complexity of this algorithm is $O(tm^2)$.

The Derived Expert-Game for This Problem is Concave.

Define a value function for the expert game by defining $\mathbf{x}$ as follows :

$$\forall S \subseteq \xi \, . \, \mathbf{x}(S) = -\hat{d}_{\xi}(S) + C,$$

where C is a normalizing constant that guarantees concavity. We assign an artificial value of C to $\hat{d}_{\xi}(\emptyset)$, thus $\mathbf{x}(\emptyset) = 0$. The proper selection of C is discussed in section 2.3.2.

We can show the concavity of this game via the condition of diminishing returns. Indeed for an $R \subseteq \xi$ let $N_R(x)$ denote the neighbours in ξ of a point x, $x \in R$. This is the set of points of ξ that are closer to x than to any other point of R.

$$N_R(x) = \{y \in \xi : x \in \arg\min_{z \in R} d_X(y, z)\}.$$

When adding x to coalition S the distance $d_X(y, S)$ may be different from the distance $d_X(y, S \cup \{x\})$ only for points $y \in N_{S \cup \{x\}}(x)$. The set of such points subsides as the base coalition grows

$$S \subseteq T \not\ni x \;\Rightarrow\; N_{S \cup \{x\}}(x) \supseteq N_{T \cup \{x\}}(x).$$

Hence, the set of points, y, for which $d_X(y, U)$ is reduced by adding x to U for $U = S$ is a superset of such points for $U = T$. As the candidate median set U grows the distance

$d_X(y, U)$ decreases weakly for all points $y \in \xi$. That is

$$S \subseteq T \;\Rightarrow\; d_X(y, S) \geq d_X(y, T).$$

It follows that as the candidate median set grows the gain achieved by adding one more point to the set diminishes:

$$\mathbf{x}(S \cup \{x\}) - \mathbf{x}(S) \geq \mathbf{x}(T \cup \{x\}) - \mathbf{x}(T).$$

Choosing C.

The previous section proves that the expert game is concave for coalitions of size greater than or equal to 1. Note that the additive constant C cancels out in the condition of diminishing returns, and hence its value is not important. To complete the proof, we have to define $C = \hat{d}_\xi(\emptyset)$ in a way that will not violate concavity. Choosing a big C would do, but this would weaken the bound we get in section 2.3.2. The diameter of a set of points is defined as:

$$\operatorname{diam} S = \sup\{d_X(x, y) : x, y \in S\}.$$

Let $C = 2 \cdot \operatorname{diam} \xi$, as

$$\max\{\hat{d}_\xi(\{x\}) : x \in \xi\} \leq \operatorname{diam} \xi$$

and

$$\max\{\hat{d}_\xi(S) - \hat{d}_\xi(S \cup \{x\}) : S \subseteq \xi, x \in \xi\} \leq \operatorname{diam} \xi$$

guarantee together that $\forall x \in \xi$, $\forall S \subseteq \xi \setminus \{x\}$:

$$\hat{d}_\xi(S) - \hat{d}_\xi(S \cup \{x\}) \leq \operatorname{diam} \xi \leq \hat{d}_\xi(\emptyset) - \hat{d}_\xi(\{x\}).$$

Estimating the Quality of the Approximation.

Each s-median problem is equivalent to a concave and monotone expert game. This can be used to bound the quality of approximation the greedy algorithm yields for this problem.

Notation 2.3.1 *Let* $\mathrm{D_j} = \hat{d}_\xi(S_j^{gr})$ *denote the average distance of the approximation of size* j *produced by the greedy algorithm; let* $\tilde{\mathrm{D}} = \hat{d}_\xi(S_s^{opt})$ *denote the average distance of the optimal solution of size* s*; and let* $\mathrm{P_{j,s}} = D_j/\tilde{D}$ *be the ratio between them.*

Theorem 2.2.3 gives us

$$\frac{-D_j + C}{-\tilde{D} + C} \geq 1 - e^{-j/s}.$$

Or

$$D_j \leq \tilde{D}[1 + e^{-j/s}(\frac{C}{\tilde{D}} - 1)].$$

To allow a comparison to Theorem 2.3.1 we let $\epsilon = e^{-j/s}(C/\tilde{D} - 1)$. Solving this for j we get

Theorem 2.3.2 *Given any* $\epsilon > 0$*, the greedy algorithm outputs a set* U *of size*

$$s(\ln\frac{1}{\epsilon} + \ln(\frac{2\operatorname{diam}\xi}{\tilde{D}} - 1)) \leq s\ln(\frac{2\operatorname{diam}\xi}{\epsilon\tilde{D}})$$

such that

$$\hat{d}_\xi(U) \leq (1+\epsilon)\tilde{D},$$

where $\tilde{D}$ *is the average distance of the optimal solution for the discrete* s*-median problem.*

Linear Programming vs. Greedy.

To compare Lin and Vitter's s-median approximation algorithm to the greedy algorithm described in this section note that their theorem gives a uniform bound for all input graphs satisfying $|V| \leq m$, while our bound is uniform for graphs with identical $\frac{\tilde{D}}{\operatorname{diam}\xi}$ ratio. The greedy algorithm's performance grows logarithmically rather than linearly with $\frac{1}{\epsilon}$. Easy implementation is another potential advantage of a vanilla greedy approach. Lin and Vitter express the quality of approximation in terms of the optimal fractional solution, while Theorem 2.3.2 expresses the quality of approximation in terms of the optimal integral solution. Cornuejols et al. [21] and Nemhauser et al. [52] show the bounds of Theorem 2.2.3 hold relative to the optimal fractional solution of the linear programming formulation of the s-median problem for $j = k$ and $\alpha = 1$. Similarly, $\tilde{D}$ can be replaced by $\hat{D}$ in Theorem 2.3.2.

Contributions of This Work to the Analysis of the Greedy Heuristic's Performance.

Previous results on the performance of the greedy heuristic of Cornuejols et al. [21] and Nemhauser et al. [52] do not allow its comparison to the Lin-Vitter approximation algorithm, as they do not consider a relaxation of the requirement on the desired size of the approximating set. Their analysis bounds only the ratio we denoted $P_{k,k}$ and not the more general $P_{j,k}$.

Yet another novelty of our work is the proof presented in this section. Historically, Cornuejols et al. derived a lower bound on the quality of a greedy approximation to the solution of an s-median problem, which was subsequently generalized to arbitrary concave and monotone games by Nemhauser et al. Our proof, by contrast, proceeds from the general to the specific.

2.4 Application to Memory-Based Learning

Having analyzed the performance of a greedy alternative to the approximation algorithm Lin and Vitter present for the s-median problem, this section compares the performance of the two approximation algorithms as tools in the construction of Voronoi Systems that model Lipschitz functions. It begins with a review of the learning algorithm. Then it compares the size of the Voronoi system required by the original algorithm of Lin and Vitter to that required by the greedy alternative analyzed in the previous section for the same user-specification of accuracy and confidence. It concludes with a review of the proof that the Lin Vitter algorithm indeed works (with either approximation subroutine).

2.4.1 The Learning Algorithm

Lin and Vitter [42] propose to learn classes of uniformly Lipschitz bounded functions by Voronoi systems of polynomial size with respect to the the error measure

$$er_{P_X}(f,g) \quad = \quad \mathbf{E_X}[d_Y(g(x),f(x))]$$

$$= \int_X d_Y(g(x), f(x))dP_X.$$

Let $Q_{P_x}(X, \epsilon, d_X)$ denote the quantization number defined to be the smallest integer s such that there exists a Voronoi encoder γ of size s that satisfies $\mathbf{E}[d_X(x, u_{\gamma(x)})] \leq \epsilon$. The algorithm draws

$$m = \Omega(\frac{s \dim X \operatorname{diam} Y}{\epsilon} \log s \log \frac{\operatorname{diam} Y}{\epsilon} + \frac{\operatorname{diam} Y}{\epsilon} \log \frac{1}{\delta}) \tag{2.3}$$

examples, where $s = Q_{P_x}(X, \frac{\epsilon}{4K}, d_X)$. It runs an s-median approximation algorithm on the sample that was drawn. The resulting median set is used to build a Voronoi system, which is output by the algorithm.

Section 2.4.3 reviews the proof that for any given ϵ, δ and any target function f in the class the algorithm outputs a Voronoi system which implements a function h for which with confidence of at least $1 - \delta$

$$er_{P_X}(f, h) \leq \epsilon.$$

2.4.2 Comparing the Two s-Median Approximation Subroutines

The size of a Voronoi system produced by the Lin-Vitter approximation algorithm is

$$\Theta(\frac{sK \cdot \operatorname{diam} Y \log m}{\epsilon}). \tag{2.4}$$

If a priori information on the distribution of the input points is available, a lower bound $d \leq \tilde{D}$ on $\tilde{D}$ may hold almost everywhere, that is for all of the space except, possibly, for a set of measure zero. For example, for m input points drawn from the uniform distribution on a region of area A in the plane with probability one the value of a solution to the s-median problem is lower bounded by $\beta(m-s)\sqrt{\frac{A}{s}}$, for some constant β, as shown by Fisher and Hochbaum [26]. Then the vanilla greedy algorithm may be used to produce a system of size $\Theta(s \cdot \log \frac{K \cdot \operatorname{diam} Y}{d\epsilon})$.

Since a confidence parameter is inherent in the evaluation of the performance of PAC learning systems, the following simpler analysis suffices for a better comparison of the two approximation schemes in the context of learning. The size of $\tilde{D}$ is lower bounded by the distance between the two nearest points in ξ. For an ordered set of m points drawn independently with respect to P_X let N_X^m denote the distance between the first point and its nearest neighbour in the set

$$N_X^m = \min\{d_X(x_1, x_i) : i = 2, 3, \ldots, m\}.$$

Consider, for example the case of a region $X \subseteq \Re^n$ where d_X is defined to be an l_P norm $|x| = (\sum_1^n x_i^p)^{1/p}$, and P_X a bounded density such that $P_X < P$. Then

$$\Pr\{N_X^m < \epsilon\} \leq Pm(2\epsilon)^n.$$

Now

$$\begin{aligned}
\Pr\{\tilde{D} < \epsilon\} &< m \Pr\{\tilde{D} < \epsilon \ \wedge \ x_1 \in \arg\min_{\{x_i \neq x_j\}} d_X(x_i, x_j)\} \\
&\leq m \Pr\{N_X^m < \epsilon\} \\
&\leq Pm^2(2\epsilon)^n.
\end{aligned}$$

Thus, for a given $\delta > 0$, with probability at least $1 - \delta$,

$$\tilde{D} \geq \frac{1}{2}(\frac{\delta}{Pm^2})^{\frac{1}{n}}.$$

Since any two norms $|.|_1, |.|_2$ on $\Re^n$ are equivalent, that is $a|x|_1 \leq |x|_2 \leq b|x|_1$ for some positive constants a, b [45], for any norm on $\Re^n$:

$$\tilde{D} \geq C(\frac{\delta}{Pm^2})^{\frac{1}{n}},$$

for some $C > 0$. For distribution-metric pairs, $\langle P_X, d_X\rangle$, for which the bound

$$\tilde{D} = \Omega((\frac{\delta}{m^k})^{\frac{1}{n}}). \tag{2.5}$$

holds with confidence $1 - \delta$, that is for all but a share δ of $\langle P_X^m, d_X^m\rangle$, a greedily chosen memory-based learning system of size

$$\Theta(s \cdot (\log \frac{K \cdot \operatorname{diam} Y}{\epsilon} + \frac{1}{\dim X} \log \frac{m^k}{\delta})) \tag{2.6}$$

can meet prespecified accuracy and confidence bounds given by parameters of ϵ and $1-\delta$. To achieve this we choose m, as specified in (2.3), for a confidence parameter of $1-\delta/2$. We also choose the size of the greedily selected approximating set specified in (2.6) with respect to confidence $1-\delta/2$. The asymptotic size of the memory-based learning system is then given by (2.6). This is a smaller system than that produced by the algorithm proposed by Lin and Vitter, the size of which is given by (2.4).

2.4.3 How to Prove That it Works

This section gives an outline of Lin and Vitter's correctness proof for the learning algorithm described in Section 2.4.1.

First we quote two definitions after Haussler [34, 35].

For $r \in \Re$ let $\operatorname{sign}(r) = 1$ iff $r > 0$, and zero otherwise.

Definition 2.4.1 *For $A \subset \Re^m$ say A is* full *if there exists an $x \in \Re^m$ such that the set of sign vectors of the following sums is of the maximum size possible*

$$|\{\ \langle \operatorname{sign}(x_i + y_i)\rangle_{i=1}^m\ : y \in A\}| = 2^m.$$

Definition 2.4.2 *Let $\mathcal{F}$ be a class of functions from a set X into $\Re$. For any sequence $\xi_X = (x_1, \ldots, x_m)$ of points in X, let $\mathcal{F}(\xi_X) = \{(f(x_1), \ldots, f(x_m)) : f \in \mathcal{F}\}$. If $\mathcal{F}(\xi_X)$ is full we say that ξ_X is* shattered *by $\mathcal{F}$. The* pseudo-dimension *of $\mathcal{F}$ denoted by $\mathbf{dim_P}\mathcal{F}$, is the largest m such that there exists a sequence of m points in X that is shattered by $\mathcal{F}$. If arbitrarily long sequences are shattered, then $\mathbf{dim_P}\mathcal{F}$ is infinite.*

If $\mathcal{F}$ is a class of $\{0, 1\}$-valued functions then the definition of the pseudo-dimension is the same as that of the VC dimension. Haussler and Long [33] showed an upper bound on the sample complexity required to guarantee the uniform convergence with confidence $1-\delta$

of the empirical estimates of a given family of functions with a bounded pseudo-dimension. Lin and Vitter show that the pseudo-dimension of Voronoi encoders of size at most s is $O(\dim X \cdot s \log s)$. Note that an ϵ/K-good Voronoi encoder guarantees an ϵ-good Voronoi system, by the Lipschitz condition.

Choosing $s = Q_{P_x}(X, \frac{\epsilon}{4K}, d_X)$ they assure that there exists an $\frac{\epsilon}{4K}$-good Voronoi encoder of size s. Then by drawing a sample of the size required by Haussler and Long they guarantee that with high confidence the empirically-best Voronoi encoder of size s is $\frac{\epsilon}{2K}$ accurate. Hence a solution to the s-median problem would produce an $\frac{\epsilon}{2}$-good Voronoi system. Since a solution is generally $\mathcal{NP}$-hard to find output an approximation that yields an ϵ-good system.

2.5 Conclusion

One of the fundamental problems of AI is filtering out redundant information. Operations researchers have investigated this problem as modeled by a Coalitional Game. In this model a sufficient condition was found for the existence of a uniform bound on the performance of the greedy approximation heuristic. The same condition on the game, monotonicity and concavity, implies a uniform bound even when approximate rather than precise values of coalitions are known. An s-median problem can be mapped to a game satisfying the condition. We use this to derive bounds on the quality of a greedy approximate solution to the s-median problem. We argue that in the context of memory-based learning of Lipschitz functions the greedy approximation algorithm is an attractive alternative to the approximation technique proposed by Lin and Vitter [42].

Further exploration of the greedy heuristic as well as other simple data processing techniques may contribute, we conjecture, to a better understanding of conscious intelligence.

Chapter 3

Scapegoat Trees

3.1 Introduction

There are a vast number of schemes available for implementing a "dictionary" – supporting the operations INSERT, DELETE, and SEARCH – using balanced binary search trees. Mehlhorn and Tsakalidis [49] survey the recent literature on such data structures. In this paper we propose a new method that achieves optimal amortized costs for update operations (INSERT and DELETE) and optimal *worst-case* cost for SEARCH, *without* requiring the extra information (e.g. colors or weights) normally required by many balanced-tree schemes. This is the first method ever proposed that achieves a worst-case search time of $O(\log n)$ without using such extra information, while maintaining optimal amortized update costs. In addition, the method is quite simple and practical.

In their comparative study Baer and Schwab [7], distinguish *height-balanced* schemes from *weight-balanced* schemes based on the criterion that triggers restructuring.

In a *height-balanced* structure the extra information stored at each node helps to enforce a bound on the overall height of the tree by bounding the height of subtrees. *Red-black trees*, were invented by Bayer [9] and refined by Guibas and Sedgewick [29]. They are an elegant example of the height-balanced approach. Red-black trees implement the basic dictionary operations with a *worst-case* cost of $O(\log n)$ per operation, at the cost of storing one extra bit (the "color" of the node) at each node. AVL trees [1] are another well-known example of height-balanced trees.

Other schemes are *weight-balanced* in that the size of subtrees causes restructuring. By ensuring that the weights of siblings are approximately equal, an overall bound on the height of the tree is enforced. Nievergelt and Reingold [53] introduce such trees and present algorithms for implementing the basic dictionary operations in $O(\log n)$ worst-case time.

The first published data structure that does not store any extra information at each node are Splay trees due to Sleator and Tarjan [64]. They achieve $O(\log n)$ amortized complexity per operation. However, splay trees do not guarantee a logarithmic worst-case bound on the cost of a SEARCH, and require restructuring even during searches (unlike scapegoat trees, which do have a logarithmic worst-case cost of a SEARCH and do not restructure the tree during searches). Splay trees do have other desirable properties that make them of considerable practical and theoretical interest, however, such as their near-optimality when handling an arbitrary sequence of operations.

Our algorithm modifies the weight-balanced method of Varghese [20, Problem 18-3], who presents an algorithm for maintaining weight-balanced trees with *amortized* cost $O(\log n)$ per operation. Our scheme combines the notions of height-balance and weight-balance to achieve an effective algorithm, without storing either height information or weight information at any node. It is most similar to Andersson's $GB_0(c)$ trees [3]. His first publication [2] has shortly preceded our independent discovery.

Both $GB_0(c)$ trees and scapegoat trees use total rebuilding of subtrees to enforce an upper bound on the depth of the tree, and achieve the same asymptotic performance for the dictionary operations. Both schemes require no balancing information to be kept at the nodes. Andersson's restructuring is triggered by a height condition. We have rediscovered his restructuring scheme, yet we also present a more general weight-based condition. The maintenance algorithm for scapegoat trees, like that for $GB_0(c)$ trees, occasionally rebuilds the whole tree to preserve the depth guarantee in the face of deletions. The condition used in scapegoat trees to trigger restructuring of the whole tree is advantageous in that it requires less frequent restructuring to enforce the same depth bound.

Yet another advantage of our scheme is demonstrated by the following scenario based on a true story. Consider a company, ComputerPeak Inc., that uses a plain binary search trees' algorithm for its small data bases. One day a decision is made to upgrade the unbalanced trees' approach. Using scapegoat trees the upgrade can be carried out without changing the format of the data, and without throwing out old code. The old code can be used as a subroutine of the novel scapegoat structure. Although this scenario may not be very likely, the same property of our data structure can prove useful in their initial coding. It suggests a natural break-up of the code's development into two phases, the first of which produces code that supports all of the system's features except performance.

We show scapegoat balancing can be used for a variety of tree-based data structures : Bentley's [12] $k-d$ trees, Leuker's [40] trees for orthogonal queries. Finkel and Bentley's [25] quad trees. For all of these a method of balancing that does not resort to extraneous balancing information at the nodes was not previously known.

We include the first experimental study of a tree-based data structure that maintains balance by partial rebuilding without storing auxiliary information at the nodes. Our experimental results suggest how scapegoat trees can be tuned for optimal performance in practice. We also compare them to other tree-based solutions of the dictionary problem. Scapegoat trees show performance superior to splay trees and for some inputs even to the more conventional red-black trees which store auxiliary balancing information at every node.

Section 3.2 introduces the basic scapegoat data structure, and some notations. Section 3.4 describes the algorithm for maintaining scapegoat trees and outlines the proof of their features. Section 3.5 proves the complexity claims. Section 3.6 describes an algorithm for rebuilding a binary search tree in linear time and logarithmic space. In Section 3.7 we show how our techniques can be used in three known multi-key tree-based data structures, and state weak conditions that suffice to allow its application to other data structures. In Section 3.7.2 we show how an existing binary search trees' data base can be upgraded to a scapegoat trees' data base without modifying data format while reusing existing code. Section 3.9 includes a detailed comparison of Andersson's $GB_0(c)$ trees to scapegoat trees. Section 3.10 reports the results of experimental evaluation of scapegoat trees. We compare a few variants of the scapegoat algorithm to each other and also compare it to other algorithms for maintenance of binary search trees. Finally, Section 3.11 concludes with some discussion and open problems.

3.2 Notations

In this section we describe the data structure of a scapegoat tree. Basically, a scapegoat tree consists of an ordinary binary search tree, with two extra values stored at the root.

Each node x of a scapegoat tree maintains the following attributes:

- $key[x]$ – The key stored at node x.
- $left[x]$ – The left child of x.
- $right[x]$ – The right child of x.

We'll also use the notations:

- $size(x)$ – the size of the sub-tree rooted at x (i.e., the number of keys stored in this sub-tree including the key stored at x).
- $brother(x)$ – the brother of node x; the other child of x's parent or NIL.
- $h(x)$ and $h(T)$ – height of a node and a tree respectively. The height of a node is the length of the longest path from that node to a leaf. The height of a tree is the height of its root.
- $d(x)$ – depth of node x. The depth of a node is the length (number of edges) of the path from the root to that node. (The root node is at depth 0.)

Note that values actually stored as fields in a node are used with brackets, whereas values that are computed as functions of the node use parentheses; each node only stores three values: *key*, *left*, and *right*. Computing $brother(x)$ requires knowledge of x's parent. Most importantly, $size(x)$ is *not* stored at x, but can be computed in time $O(size(x))$ as necessary.

The tree T as a whole has the following attributes:

- $root[T]$ – A pointer to the root node of the tree.
- $size[T]$ – The number of nodes in the tree. This is the same as $size(root[T])$. In our complexity analyses we also denote $size[T]$ by n.
- $max\_size[T]$ – The maximal value of $size[T]$ since the last time the tree was completely rebuilt. If DELETE operations are not performed, then the *max_size* attribute is not necessary.

3.3 Preliminary Discussion

SEARCH, INSERT and DELETE operations on scapegoat trees are performed in the usual way for binary search trees, except that, occasionally, after an update operation (INSERT or DELETE) the tree is restructured to ensure that it contains no "deep" nodes.

A binary-tree node x is said to be **α-weight-balanced**, for some α, $1/2 \leq \alpha < 1$, if both

$$\begin{aligned} \text{size}(\text{left}[x]) &\leq \alpha \cdot \text{size}(x), \text{ and} && (3.1) \\ \text{size}(\text{right}[x]) &\leq \alpha \cdot \text{size}(x) . && (3.2) \end{aligned}$$

We call a tree **α-weight-balanced** if, for a given value of α, $1/2 \leq \alpha < 1$, all the nodes in it are α-weight-balanced. Intuitively, a tree is α-weight-balanced if, for any subtree, the sizes of its left and right subtree are approximately equal.

We denote

$$h_\alpha(n) = \lfloor \log_{(1/\alpha)} n \rfloor,$$

and say that a tree T is **α-height-balanced** if it satisfies

$$h(T) \leq h_\alpha(n), \tag{3.3}$$

where $n = size(T)$. Intuitively, a tree is α-height-balanced if its height is not greater than that of the heighest α-weight-balanced tree of the same size. The following standard claim justifies this interpretation.

Claim 3.3.1 *If T is an α-weight-balanced binary search tree, then T is α-height-balanced.*

Although scapegoat trees are not guaranteed to be α-weight-balanced at all times, they are **loosely** α-height-balanced, in that they satisfy the bound

$$h(T) \leq h_\alpha(T) + 1, \tag{3.4}$$

where $h_\alpha(T)$ is a shorthand for $h_\alpha(size[T])$.

We assume from now on that a fixed α, $1/2 < \alpha < 1$, has been chosen. For this given α, we call a node of depth greater than $h_\alpha(T)$ a **deep** node. In our scheme the detection of a deep node triggers a restructuring operation.

3.4 Operations on Scapegoat Trees

3.4.1 Searching a Scapegoat Tree.

In a scapegoat tree, SEARCH operations proceed as in an ordinary binary search tree. No restructuring is performed.

3.4.2 Inserting into a Scapegoat Tree.

To insert a node into a scapegoat tree, we insert it as we would into an ordinary binary search tree, increment $size[T]$, and set $max\_size[T]$ to be the maximum of $size[T]$ and $max\_size[T]$. Then—if the newly inserted node is deep—we rebalance the tree as follows.

Let x_0 be the newly inserted deep node, and in general let x_{i+1} denote the parent of x_i. We climb the tree, examining x_0, x_1, x_2, and so on, until we find a node x_i that is not α-weight-balanced. Since x_0 is a leaf, $size(x_0) = 0$. We compute $size(x_{j+1})$ using the formula

$$size(x_{j+1}) = size(x_j) + size(brother(x_j)) + 1 \tag{3.5}$$

for $j = 1, 2, \ldots, i$, using additional recursive searches.

We call x_i, the ancestor of x_0 that was found that is not α-weight-balanced, the **scapegoat** node. A scapegoat node must exist, by Claim 3.5.1 below.

Once the scapegoat node x_i is found, we **rebuild** the subtree rooted at x_i. To rebuild a subtree is to replace it with a 1/2-weight-balanced subtree containing the same nodes. This can be done easily in time $O(size(x_i))$. Section 3.6 describes how this can be done in space $O(\log n)$ as well.

An alternative way to find a scapegoat node.

As can be seen in Figure 3.4.2, x_0 might have more than one weight-unbalanced ancestor. Any weight-unbalanced ancestor of x_0 may be chosen to be the scapegoat. Here we show that another way of finding a weight-unbalanced ancestor x_i of x_0 is to find the deepest

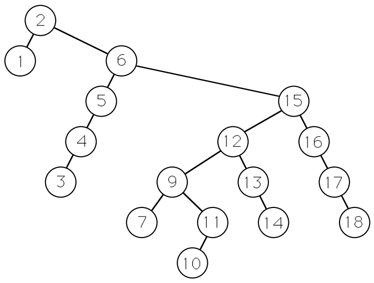

Figure 3-1: The initial tree, T. For $\alpha = 0.57$, $h_\alpha(17) = h_\alpha(18) = 5$, and T is loosely α-height-balanced (because node 10 is at depth 6). Nodes 2, 5, 6, 12, 15 and 16 are currently weight-unbalanced. Inserting 8 into this tree triggers a rebuild. We chose node 6 to be the scapegoat node.

ancestor of x_0 satisfying the condition

$$i > h_\alpha(size(x_i)). \tag{3.6}$$

Since this ancestor will often be higher in the tree than the first weight-unbalanced ancestor, it may tend to yield more balanced trees on the average. (In our experiments this heuristic performed better than choosing the first weight-unbalanced ancestor to be the scapegoat.) Inequality (3.6) is satisfied when $x_i = root[T]$, hence this scheme will always find a scapegoat node. The scapegoat node found is indeed weight-unbalanced by Claim 3.5.2.

Note that applying condition (3.6) when searching for the scapegoat in the example in Figure 3.4.2 indeed results in node 6 being rebuilt, since it is the first ancestor of node 8 that satisfies the inequality.

3.4.3 Deleting from a Scapegoat Tree.

Deletions are carried out by first deleting the node as we would from an ordinary binary search tree, and decrementing $size[T]$. Then, if

$$size[T] < \alpha \cdot max\_size[T] \tag{3.7}$$

we rebuild the whole tree, and reset $max\_size[T]$ to $size[T]$.

3.4.4 Remarks.

- Every time the whole tree is rebuilt $max\_size[T]$ is set to $size[T]$.
- Note that $h_\alpha(T)$ is easily computed from the information stored at the root. (Indeed, it could even be stored there as an extra attribute.)
- We do not need explicit *parent* fields in the nodes to find the scapegoat node, since we are just climbing back up the path we came down to insert the new node; the nodes x_i on this path can be remembered on the stack.

3.5 Correctness and Complexity

Now we prove the algorithm just described is indeed correct and analize its complexity.

3.5.1 Correctness.

The following two claims prove that the algorithm is indeed correct.

The first claim guarantees that a deep node has an ancestor that in not α-weight-balanced.

Claim 3.5.1 *If x is a node at depth greater than $h_\alpha(T)$ then there is an α-weight-unbalanced ancestor of x.*

Proof: By negation according to equations (3.1) if x is a child of y, then

$$size(x) \leq \alpha \cdot size(y).$$

By induction on the path from x to the root, $size(x) \leq \alpha^{d(x)} \cdot size[T]$. Hence, the depth $d(x)$ of a node x is at most $\log_{(1/\alpha)} size[T]$ establishing the claim. □

The following claim proves that a scapegoat node found using inequality (3.6) is weight-unbalanced.

Claim 3.5.2 *If a binary tree T contains a node x_0 at depth greater than $h_\alpha(n)$, then the deepest ancestor x_i of x_0 that is not α-height-balanced is not α-weight-balanced either.*

Proof: We chose x_i so that the following inequalities are satisfied.

$$i > h_\alpha(size(x_i)) \ ,$$

and

$$i - 1 \leq h_\alpha(size(x_{i-1})) \ .$$

Subtracting these two inequalities gives

$$\begin{aligned} 1 &> h_\alpha(size(x_i)) - h_\alpha(size(x_{i-1}) \\ &= \log_{1/\alpha}\left(\frac{size(x_i)}{size(x_{i-1})}\right) \ . \end{aligned}$$

Therefore,

$$size(x_{i-1}) > \alpha \cdot size(x_i).$$

□

3.5.2 Complexity of Searching.

Since a scapegoat tree is loosely α-height-balanced and α is fixed, a SEARCH operation takes *worst-case* time

$$O(h_\alpha(n)) = O(\log n) \ .$$

No restructuring or rebalancing operations are performed during a SEARCH. Therefore, not only do scapegoat trees yield an $O(\log n)$ worst-case SEARCH time, but they should also be efficient in practice for SEARCH-intensive applications since no balancing overhead is incurred for searches.

3.5.3 Complexity of Inserting.

The following claim is key to the complexity analysis.

Claim 3.5.3 *The time to find the scapegoat node x_i is $O(size(x_i))$.*

Proof: The dominant part of the cost of finding the scapegoat node x_i is the cost of computing the values $size(x_0)$, $size(x_1)$, $\ldots$, $size(x_i)$. Observe that with the optimized *size* calculations described in equation (3.5), each node in the subtree rooted at the scapegoat node x_i is visited exactly once during these computations. □

We now analyze the situation where no DELETE operations are done; only INSERT and SEARCH operations are performed. The following claims yield Theorem 3.5.1, which shows that a scapegoat tree is always α-height-balanced if no deletions are performed. The next claim asserts that rebuilding a tree does not make it deeper.

Claim 3.5.4 *If T is a $1/2$-weight-balanced binary search tree, then no tree of the same size has a smaller height.*

Proof: Straightforward. □

Claim 3.5.5 *If the root of T is not α-weight-balanced then its heavy subtree contains at least 2 nodes more than its light subtree.*

Proof: Denote by s_h and s_l the sizes of the heavy and the light subtrees respectively. The root of the tree is not α-weight-balanced, hence:

$$s_h > \alpha \cdot (s_h + s_l + 1)$$

This yields:

$$s_h > \frac{\alpha}{1-\alpha} \cdot (s_l + 1)$$

Since $\alpha > 1/2$ and s_h and s_l are both whole numbers, we get:

$$s_h \geq s_l + 2 \; .$$

□

A tree T is **complete** of height h if a node cannot be added to T without making its height greater than h. A complete tree of height h has $2^{h+1} - 1$ nodes.

Claim 3.5.6 *If T is not α-weight-balanced and T contains only one node at depth $h(T)$ then rebuilding T decreases its height.*

Proof: Let x be the deepest node of T, and let T_l be the light subtree of T. Let T_l' be the tree we get by removing x from T_l if x is a node of T_l, or T_l itself if x is not a node of T_l. By Claim 3.5.5, T_l' is not a complete tree of height $h(T)-1$. Therefore, Claim 3.5.4 completes the proof. □

Theorem 3.5.1 *If a scapegoat tree T was created from a 1/2-weight-balanced tree by a sequence of* Insert *operations, then T is α-height-balanced.*

Proof: By induction on the number of insert operations using Claim 3.5.6. □

Let us now consider a sequence of n Insert operations, beginning with an empty tree. We wish to show that the amortized complexity per Insert is $O(\log n)$.

For an overview of amortized analysis, see Cormen et al. [20]. We begin by defining a nonnegative *potential function* for the tree. Let

$$\Delta(x) = |size(left[x]) - size(right[x])|,$$

and define the potential of node x to be 0 if $\Delta(x) < 2$, and $\Delta(x)$ otherwise. The potential of a 1/2-weight-balanced node is thus 0, and the potential of a node x that is not α-weight-balanced is $\Theta(size(x))$. (Note that $\Delta(x)$ is not stored at x nor explicitly manipulated during any update operations; it is just an accounting fiction representing the amount of "prepaid work" available at node x.) The potential of the tree is the sum of the potentials of its nodes.

It is easy to see that by increasing their cost by only a constant factor, the insertion operations that build up a scapegoat tree can pay for the increases in potential at the nodes. That is, whenever we pass by a node x to insert a new node as a descendant of x, we can pay for the increased potential in x that may be required by the resulting increase in $\Delta(x)$.

The potential of the scapegoat node, like that of any non-α-weight-balanced node, is $\Theta(size(x_i))$. Therefore, this potential is sufficient to pay for finding the scapegoat node and rebuilding its subtree. (Each of these two operations has complexity $\Theta(size(x_i))$.) Furthermore, the potential of the rebuilt subtree is 0, so the entire initial potential may be used up to pay for these operations. This completes the proof of the following theorem.

Theorem 3.5.2 *A scapegoat tree can handle a sequence of n* Insert *and m* Search *operations, beginning with a 1/2-weight-balanced tree. with $O(\log n)$ amortized cost per* Insert *and $O(\log k)$ worst-case time per* Search, *where k is the size of the tree the* Search *is performed on.*

3.5.4 Complexity of Deleting.

The main claim of this section, Claim 3.5.10, states that scapegoat trees are loosely α-height-balanced (recall inequality (3.4)). Since we perform $\Omega(n)$ operations between two successive rebuilds due to delete operations we can "pay" for them in the amortized sense. Therefore, combining Claim 3.5.10 with the preceding results completes the proof of the following theorem.

Theorem 3.5.3 *A scapegoat tree can handle a sequence of n* Insert *and m* Search *or* Delete *operations, beginning with a 1/2-weight-balanced tree, with $O(\log n)$ amortized cost per* Insert *or* Delete *and $O(\log k)$ worst-case time per* Search, *where k is the size of the tree the* Search *is performed on.*

The first claim generalizes Theorem 3.5.1.

Claim 3.5.7 *For any tree T let $T' =$* Insert(T, x), *then*

$$h(T') \leq \max(h_\alpha(T'), h(T)) \ .$$

Proof: If the insertion of x did not trigger a rebuild, then the depth of x is at most $h_\alpha(T')$ and we are done.

Otherwise, suppose x was initially inserted at depth d in T, where $d > h_\alpha(T')$, thereby causing a rebuild. If T already contained other nodes of depth d we are done, since a rebuild does not make a tree deeper. Otherwise, the arguments in section 3.5.1 and Claim 3.5.6 apply. □

Claim 3.5.8 *If $h_\alpha(T)$ does not change during a sequence of* INSERT *and* DELETE *operations then* $\max(h_\alpha(T), h(T))$ *is not increased by that sequence.*

Proof: A DELETE operation can not increase $\max(h_\alpha(T), h(T))$. For an INSERT we have

$$h(T') \leq \max(h_\alpha(T'), h(T))$$

by Claim 3.5.7. Hence

$$\max(h_\alpha(T'), h(T')) \leq \max(h_\alpha(T'), h(T)) = \max(h_\alpha(T), h(T)) .$$

The claim follows by induction on the number of operations in the sequence. □

Claim 3.5.9 *For $T' = $* INSERT$(T, x)$*, if T is loosely α-height-balanced but is not α-height-balanced, and $h_\alpha(T') = h_\alpha(T) + 1$, then T' is α-height-balanced.*

Proof: We know that

$$h(T) = h_\alpha(T) + 1.$$

Hence

$$h(T) = h_\alpha(T').$$

Combining this with Claim 3.5.7 gives

$$h(T') \leq h_\alpha(T') ,$$

i.e., T' is height balanced. □

Now we have the tools to prove the main claim of this section.

Claim 3.5.10 *A scapegoat tree built by* INSERT *and* DELETE *operations from an empty tree is always loosely α-height-balanced.*

Proof: Let $o_1, \ldots, o_n$ be a sequence of update operations that is applied to a 1/2-weight-balanced scapegoat tree, up until (but not including) the first operation, if any, that causes the entire tree to be rebuilt. To prove the claim it suffices to show that during this sequence of operations the tree is always loosely α-height-balanced. During any sequence of update operations that do not change $h_\alpha(T)$, a loosely α-height-balanced tree remains loosely α-height-balanced, and an α-height-balanced tree remains α-height-balanced, by Claim 3.5.8. Therefore, let $o_{i_1}, \ldots, o_{i_k}$ be the subsequence (not necessarily successive) of operations that change $h_\alpha(T)$. An INSERT operation in this subsequence leaves the tree α-height-balanced, by Claim 3.5.9. The usage of $max\_size[T]$ in DELETE implies that there are no two successive DELETE operations in this subsequence, since the entire tree would be rebuilt no later than the second such DELETE operation. Therefore a DELETE operation in this subsequence must operate on an α-height-balanced tree. Since the DELETE operation decreases $h_\alpha(T)$ by just one, the result is a loosely α-height-balanced tree. The claim follows from applying the preceding claims in an induction on the number of operations. □

Proof (of Theorem 3.5.3): The proof of Theorem 3.5.1 can be easily modified to show that the amortized complexity of INSERTing and DELETEing is logarithmic. That is the potential saved at the scapegoat node can "pay" the cost of rebuilding and possibly searching in the amortized sense. A similar argument to that in the proof of Theorem 3.5.1 holds for DELETE triggered rebuilding of the root.

By Claim 3.5.10 the height of a scapegoat tree is always logarithmic in the number of nodes. Thus accounting for the worst-case performance of SEARCHes claimed in the theorem. □

3.6 Rebuilding in Place

A straightforward way of rebuilding a tree is to use a stack of logarithmic size to traverse the tree in-order in linear time and copy its nodes to an auxiliary array. Then build the new 1/2-weight-balanced tree using a "divide and conquer" method. This yields $O(n)$ time and space complexity. Chang and Iyengar [31] survey a few techniques for rebuilding trees using logarithmic auxiliary space, and present additional algorithms.

The algorithms we present in this section are not included in their survey. All of the algorithms they present require two traversals of the tree. The non-recursive technique presented in Section 3.6.2 takes advantage of the fact that the input subtree is known to be of depth logarithmic in the size of the whole tree to perform rebuilding in a single pass.

3.6.1 A Simple Recursive Method.

The first algorithm links the elements together into a list, rather than copying them into an array.

The initial tree-walk is implemented by the following procedure, FLATTEN. A call of the form FLATTEN(x, NIL) returns a list of the nodes in the subtree rooted at x, sorted in nondecreasing order. In general, a call of the form FLATTEN(x, y) takes as input a pointer x to the root of a subtree and a pointer y to the first node in a list of nodes (linked using their *right* pointer fields). The set of nodes in the subtree rooted at x and the set of nodes in the list headed by y are assumed to be disjoint. The procedure returns the list resulting from turning the subtree rooted at x into a list of nodes, linked by their *right* pointers, and appending the list headed by y to the result.

FLATTEN(x, y)

if $x =$ NIL
    then return y
right[x] $\leftarrow$ FLATTEN(*right*[x], y)
return FLATTEN(*left*[x], x)

The procedure runs in time proportional to the number of nodes in the subtree, and in space proportional to its height

The following procedure, BUILD-TREE, builds a 1/2-weight-balanced tree of n nodes from a list of nodes headed by node x. It is assumed that the list of nodes has length at least $n + 1$. The procedure returns the $n + 1$st node in the list, s, modified so that *left*[s] points to the root r of the n-node tree created.

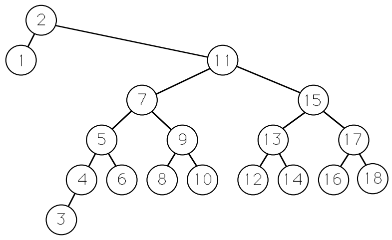

Figure 3-2: The tree INSERT$(T, 8)$, where T is the tree of Figure 1.

BUILD-TREE(n, x)
if $n = 0$
  then *left*$[x] \leftarrow$ NIL
    return x
$r \leftarrow$ BUILD-TREE$(\lceil (n-1)/2 \rceil, x)$
$s \leftarrow$ BUILD-TREE$(\lfloor (n-1)/2 \rfloor, right[r])$
right$[r] \leftarrow$ *left*$[s]$
left$[s] \leftarrow r$
return s

A call to BUILD-TREE$(n, scapegoat)$ runs in time $O(n)$ and uses $O(\log n)$ space.

The following procedure, REBUILD-TREE, takes as input a pointer *scapegoat* to the root of a subtree to be rebuilt, and the size n of that subtree. It returns the root of the rebuilt subtree. The rebuilt subtree is 1/2-weight-balanced. The procedure utilizes the procedures FLATTEN and BUILD-TREE defined above, and runs in time $O(n)$ and space proportional to the height of the input subtree.

REBUILD-TREE$(n, scapegoat)$
create a dummy node w
$z \leftarrow$ FLATTEN$(scapegoat, w)$
BUILD-TREE(n, z)
return *left*$[w]$

Figures 3.4.2 and 3 illustrate this process.

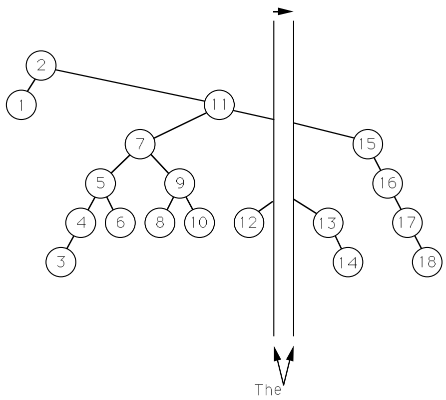

Figure 3-3: Non-recursive rebuilding in place. An intermediate state during the execution of a rebuilding in place of the tree INSERT$(T, 8)$. Node 11 is the new root of the subtree being rebuilt. (See T in Figure 1).

3.6.2 A Non-Recursive Method.

This section suggests a non-recursive method for rebuilding a tree in logarithmic space, that proved to be faster in our experiments than the previous version.

We traverse the old tree in-order. Since the number of nodes in the tree is known, the new place of each node we encounter can be uniquely determined. Every node is "plugged into" the right place in the new tree upon being visited, thereby creating the new tree in place.

We need to keep track of the "cutting edge" of the two tree traversals as shown in Figure 3.6.2. Since the depth of both trees is logarithmic, two logarithmic size stacks suffice for this purpose.

The procedure REBUILD-TREE provides the same interface as the procedure with the same name given in sub-section 3.6.1. It calls the procedures GET-NEXT-NODE and ADD-NEW-NODE, which are described below.

Our pseudo-code calls the standard stack-handling routines POP, PUSH, CREATE and TOP. It also uses SECOND – a routine that peeks at the second element on the stack.

```
Rebuild-Tree(n, scapegoat)
insert_type ← I-TYPE-LEFT
slots_in_last_level ← 2^⌊lg n⌋
nodes_for_last_level ← n − slots_in_last_level + 1
ratio ← nodes_for_last_level/slots_in_last_level
Create(Ruining_Stack)
Create(Building_Stack)
Push(Ruining_Stack, scapegoat)
while n > 0
     do n ← n − 1
        insert_type ← Add-New-Node(Get-Next-Node(), insert_type)
return Top(Building_Stack)
```

The routine Get-Next-Node traverses the old tree in-order. It uses a stack – *Ruining_Stack* – to store pointers to the nodes of the old subtree. The size of this stack is bound by the depth of the subtree being rebuilt, i.e. by $h_\alpha(n) + 2$, where n is the size of the subtree.

```
Get-Next-Node()
next_node ← Top(Ruining_Stack)
while left[next_node] ≠ NIL
     do father_node ← next_node
        next_node ← left[next_node]
if next_node = Top(Ruining_Stack)
   then Pop(Ruining_Stack)
   else left[father_node] ← NIL
if right[next_node] ≠ NIL
   then Push(Ruining_Stack, right[next_node])
return next_node
```

The routine Add-New-Node creates a perfectly balanced tree from the nodes that are passed to it in-order.

ADD-NEW-NODE accesses and modifies the global variables *ratio*, *nodes_for_last_level* and *slots_in_last_level* that were set by REBUILD-TREE. It assumes that the number of times it will be called is compatible with the initial value of *nodes_for_last_level*.

The parameters of ADD-NEW-NODE are *next_node* and *insert_type*. The first – *next_node* – is a tree-node. The nodes are assumed to be passed in-order. The second parameter – *insert_type* – can be equal to one of three constants: I-T-LEFT – if the node is to be inserted as a leaf which is a left son of its parent; I-T-RIGHT – same as I-T-LEFT only for a right son; and I-T-PARENT – if the node is not a leaf. ADD-NEW-NODE returns the value that should be passed as *insert_type* on the next call.

ADD-NEW-NODE uses a stack – *Building_Stack* – the size of which is bound by $\lg n+1$, where n is the size of the subtree being rebuilt. The records stored on this stack contain four fields – a pointer to a tree node, *height*, *lacks_right_son* and *lacks_father*. The *height* field is a positive integer that records the height of the appropriate node in the new tree relatively to the deepest leaf in the tree. The boolean fields *lacks_right_son* and *lacks_father* indicate the reason that caused us to push the record on the stack. Possible reasons are – the node does not have a father yet, or the node's right son was not determined yet. For every record on the stack at least one of *lacks_right_son* and *lacks_father* is set to TRUE. We will refer to these fields in the order in which they were described. Hence, $\{node, 7, \text{TRUE}, \text{FALSE}\}$ will denote a record that points to node *node*, with *height* equal to 7, *lacks_right_son* set to TRUE and *lacks_father* set to FALSE.

```
ADD-NEW-NODE(next_node, insert_type)
if insert_type ≠ I-T-PARENT
   then slots_in_last_level ← slots_in_last_level − 1
        if nodes_for_last_level/slots_in_last_level < ratio
           then return SKIP-A-LEAF(next_node, insert_type)
           else nodes_for_last_level ← nodes_for_last_level − 1
                return ADD-A-LEAF(next_node, insert_type)
   else return ADD-NON-LEAF(next_node)
```

```
Skip-A-Leaf(next_node, insert_type)
if insert_type = I-T-LEFT
   then skip a left leaf
        left[next_node] ← NIL
        if height[Top(Building_Stack)] = 2
           then right[Top(Building_Stack)] ← next_node
                if ¬lacks_father[Top(Building_Stack)]
                   then Pop(Building_Stack)
                   else lacks_right_son[Top(Building_Stack)] ← FALSE
                Push(Building_Stack, {next_node, 1, TRUE, FALSE})
           else Push(Building_Stack, {next_node, 1, TRUE, TRUE})
        return I-T-RIGHT
   else skip a right leaf
        right[Top(Building_Stack)] ← NIL
        if ¬lacks_father[Top(Building_Stack)]
           then Pop(Building_Stack)
           else lacks_right_son[Top(Building_Stack)] ← FALSE
        return Add-Non-Leaf(next_node)
```

```
Add-A-Leaf(next_node, insert_type)
right[next_node] ← NIL
left[next_node] ← NIL
if insert_type = I-T-LEFT
   then Push(Building_Stack, {next_node, 0, FALSE, TRUE})
   else  right[Top(Building_Stack)] ← next_node
         if lacks_father[Top(Building_Stack)]
            then lacks_right_son[Top(Building_Stack)] ← FALSE
            else  Pop(Building_Stack)
return I-T-PARENT
```

```
Add-Non-Leaf(next_node)
left[next_node] ← Top(Building_Stack)
next_node's_height ← height[Top(Building_Stack)] + 1
Pop(Building_Stack)
if height[Second(Building_Stack)] = next_node's_height + 1
   then right[Top(Building_Stack)] ← next_node
        if ¬lacks_father[Top(Building_Stack)]
           then Pop(Building_Stack)
           else  lacks_right_son[Top(Building_Stack)] ← FALSE
        Push(Building_Stack, {next_node, next_node's_height, TRUE, FALSE})
   else  Push(Building_Stack, {next_node, next_node's_height, TRUE, TRUE})
if height[Top(Building_Stack)] > 1
   then return I-T-LEFT
   else  return I-T-RIGHT
```

3.7 More on Applications of Scapegoat Techniques

Scapegoat balancing techniques are applicable not only to binary search trees, but also to other tree-based data structures. We first state sufficient conditions for their applicability, and then describe some known data structures which meet these conditions. No balancing scheme that does not store auxiliary information at the nodes was previously known for the multi-key data structures to which we show scapegoat techniques can be applied. The discussion in the second subsection, addresses the upgrading of code that supports unbalanced data structures to scapegoat-balanced structures. It also suggests steps for the initial coding of scapegoat trees.

3.7.1 Multi-Key Data

The ideas underlying scapegoat trees are that of finding and rebuilding a subtree whose root is not weight-balanced when the tree gets too deep, and periodically rebuilding the root after enough DELETEs occurred. This technique can be applied to other tree-like data structures. To allow this, it should be possible to find the scapegoat node and to rebuild the subtree rooted at it. The time to find the scapegoat and the rebuilding time does not have to be linear in the number of nodes in the subtree being rebuilt, as was the case with binary search trees (Theorem 3.5.3). It is also not necessary for the rebuilding algorithm to yield a perfectly balanced subtree. These generalizations of the main theorem, allow us to apply scapegoat techniques to an array of other tree-like data structures.

A Stronger Version of the Main Theorem.

Suppose for a class of trees, some fixed $\alpha_{bal} \geq 1/2$ and a function F, $F(n) = \Omega(1)$, satisfying $F(Cn) = O(F(n))$ for any constant C, there exists an algorithm that when given n nodes can in $O(nF(n))$ steps build a tree containing those nodes that is α_{bal}-weight-balanced. We'll call such a rebuilding routine a α_{bal}-**relaxed** rebuilding routine. Also suppose there exists an algorithm that can find an ancestor of a given node that is not weight-balanced in $O(nF(n))$ time, where n is the size of the subtree rooted at the scapegoat node, provided such an ancestor exists. Then we can use scapegoat techniques to support dynamic updates to this class with amortized logarithmic complexity. When $F(n)$ is constant and $\alpha_{bal} = 1/2$, we have the previously handled situation of Theorem 3.5.3.

For a fixed $\alpha_{trigger}$, $\alpha_{trigger} > \alpha_{bal}$, an insertion of a deep node with respect to $\alpha_{trigger}$ would trigger a rebuilding. Claim 3.5.1 guarantees that such a node has an $\alpha_{trigger}$-weight-unbalanced ancestor. However, for any constants α, β, $1/2 < \alpha < \beta$ and for n large enough there exists a β-weight-unbalanced tree of size n that can be rebuilt into a deeper α-weight-balanced tree. Hence, we cannot choose any $\alpha_{trigger}$-weight unbalanced ancestor of the deep node to be the scapegoat. However, if we choose as a scapegoat an ancestor x of the deep node that satisfies condition (3.6):

$$h(x) > h_{\alpha_{trigger}}(size(x)), \tag{3.8}$$

we can prove the following theorem.

Theorem 3.7.1 *A relaxed scapegoat tree can handle a sequence of n* INSERT *and m* SEARCH *or* DELETE *operations, beginning with a $1/2$-weight-balanced tree, with an amortized cost of $O(F(n)\ \log_{1/\alpha_{trigger}} n)$ per* INSERT *or* DELETE *and $O(\log_{1/\alpha_{trigger}} k)$ worst-case time per* SEARCH*, where k is the size of the tree the* SEARCH *is performed on.*

Proof (sketch) : The existence of an ancestor that satisfies equation (3.8) is guaranteed as explained in Section 3.5 (the root of the tree satisfies it). It follows from the way the scapegoat was chosen that rebuilding the subtree rooted at it decreases the depth of the rebuilt subtree, allowing us to prove a result similar to Claim 3.5.7. The other claims leading to Theorem 3.5.3 can also be proven for relaxed rebuilding. Hence, we can indeed support a tree of depth at most $\log_{1/\alpha_{trigger}} k + 1$, where k is the size of the tree, thereby establishing the bound on the worst-case search time.

To prove the amortized bound on the complexity of updates we will define a potential function Φ in an inductive manner. Let the potential of the nodes in a subtree that was just rebuilt and of newly inserted nodes be 0. Every time a node is traversed by an update operation, increase its potential by $F(N)$, where N is the size of the subtree rooted at that node. For any update operation, the node whose potential is increased the most is the root.

Hence the total price of the update operation is bounded by

$$(F(N)+1)\log_{1/\alpha_{trigger}} N = O(F(N)\log_{1/\alpha_{trigger}} N)$$

as $F(n) = \Omega(1)$.

If the root is $\alpha_{trigger}$-weight unbalanced, then CN different update operations traversed it since it was inserted or last rebuilt. Now $C \geq C_0$, where

$$C_0 = \frac{\alpha_{trigger} - \alpha_{bal}}{2\alpha_{trigger}\alpha_{bal}}.$$

At each one of the last C_0 passes the potential of the root was increased by at least $F((1-C_0)N)$. Hence, the total potential stored at the root is at least $C_0 NF((1-C_0)N) = O(NF(N)$, allowing it to pay for the rebuilding operation.

□

Scapegoat $k-d$ Trees.

Bentley [12] introduced $k-d$ trees. He proved average-case bounds of $O(\lg n)$ for a tree of size n for both updates and searches. Bentley [13] and Overmars and van Leeuwen [55] propose a scheme for dynamic maintenance of $k-d$ trees that achieves a logarithmic worst-case bound for searches with an average-case bound of $O((\lg n)^2)$ for updates. Both use an idea similar to ours of rebuilding weight-unbalanced subtrees. Overmars and van Leeuwen called their structure pseudo $k-d$ trees.

Scapegoat $k-d$ trees achieve logarithmic worst-case bounds for searches and a $\log^2 n$ amortized bound for updates. (The analysis of updates of Overmars and van Leeuwen [55] and Bentley [13] can be improved to yield amortized rather than average-case bounds.) However, scapegoat $k-d$ trees do not require maintaining extra data at the nodes. Also we believe they might prove to be faster in practice as they do not rebuild every weight-unbalanced node, thereby allowing for it to become balanced by future updates.

Applying Theorem 3.7.1 we get:

Theorem 3.7.2 *A scapegoat $k-d$ tree can handle a sequence of n* Insert *and m* Search *or* DELETE *operations, beginning with a 1/2-weight-balanced tree, with* $O(\log^2 n)$ *amortized cost per* INSERT *or* DELETE *and* $O(\log k)$ *worst-case time per* SEARCH*, where* k *is the size of the tree the* SEARCH *is performed on.*

Proof: To apply Theorem 3.7.1 we use the algorithm Bentley [12] proposes for building a perfectly balanced $k-d$ tree of N nodes in $O(kN \lg N)$, by taking as a splitting point the median with respect to the splitting coordinate. Finding the scapegoat is done in a manner similar to that in binary search trees. □

Scapegoat Trees for Orthogonal Queries.

For keys which are d dimensional vectors one may wish to specify a range for each component of the key and ask how many keys have all components in the desired range. Leuker [40] proposed an algorithm that handles range queries in $O(log^d n)$ worst-case time where n is the size of the tree. Updates are handled in $O(nlog^d n)$ amortized time.

Leuker's paper proves that given a list of n keys a 1/3-balanced tree may be formed in $O(nlog^{\min(1,d-1)} n)$ time.

Using this in Theorem 3.7.1 proves

Theorem 3.7.3 *A scapegoat orthogonal tree can handle a sequence of* n INSERT *and* m SEARCH *or* DELETE *operations, beginning with a 1/2-weight-balanced tree, with* $O(\log^{\min(2,d)} n)$ *amortized cost per* INSERT *or* DELETE *and* $O(\log^d k)$ *worst-case time per range query, where* k *is the size of the tree the range query is performed on.*

Note that our algorithm improves Leuker's amortized bounds for updates, and does not require storage of balancing data at the nodes of the tree.

Scapegoat Quad Trees.

Quad trees were introduced by Finkel and Bentley [25]. They achieve a worst-case bound of $O(log_2 N)$ per search. (As in a d dimensional quad tree every node has 2^d children naively one could expect a $O(log_{2^d} N)$ worst-case search time.) They do not address deletion, and give only experimental results for insertion times. Samet [62] proposed an algorithm for deletions. Overmars and van Leeuwen [55] introduced pseudo-quad trees – a dynamic version of quad trees. They suggest an algorithm for achieving $O((\lg N)^2)$ average insertion and deletion times, where N is the number of insertions, while improving the worst-case search time to $\log_{d+1-\delta} n + O(1)$, where d is the dimension of the tree, n the size of the tree the search is performed on, and δ an arbitrary constant satisfying $1 < \delta < d$.

Scapegoat quad trees can be compared to pseudo-quad trees:

- Scapegoat trees offer worst-case search time of $C \log_{d+1} n$ for any constant C, or following the original notations of Overmars and van Leeuwen $\log_{d+1-\delta} n$ for any positive constant δ (note that we do not require $1 < \delta$).

- The bounds on updates are improved from average-case to amortized bounds. (Though careful analysis of the algorithm of Overmars and van Leeuwen [55] can yield amortized bounds too.)

- Scapegoat trees do not require maintenance of extra data at the nodes regarding the weight of the children of each node. This can be quite substantial in this case, as each node has 2^d children, where d is the dimension of the tree.

- Scapegoat trees might prove faster in practice, as they do not require the rebuilding of every weight-unbalanced node, thereby allowing some nodes to be balanced by future updates. Also more compact storage might result in greater speed.

We call a multi-way node, x, α-weight-balanced, if the every child y of x, satisfies $\text{size}(y) \leq \alpha \text{size}(x)$. Weight and height balanced trees are defined in a way similar to that used for binary trees.

Theorem 2.2.3 in Overmars and van Leeuwen [55] suggests how to build a $1/(d+1)$ weight balanced pseudo-quad tree in $O(n \log n)$ time. Finding a scapegoat in a multiway tree can be done by traversing a tree in a manner similar to that described for binary trees, starting at the deep node and going up. Plugging this into Theorem 3.7.1 proves:

Theorem 3.7.4 *A scapegoat quad tree can handle a sequence of n* INSERT *and m* SEARCH *or* DELETE *operations, beginning with a 1/2-weight-balanced tree, with $O(\log^2 n)$ amortized* *cost per* INSERT *or* DELETE *and* $O(\log_{d+1-\delta} k)$ *worst-case time per* SEARCH*, where* k *is the size of the tree the* SEARCH *is performed on.*

3.7.2 Upgrading Unbalanced Binary Search Trees to Scapegoat Trees

To upgrade an existing data base that uses a binary search trees' data representation, a change to the data itself is not required. One may continue to use the existing code to perform searches and modification of the data base. Scapegoat trees can be implemented as a software layer above the existing code that uses the existing code as a subroutine.

The scapegoat layer can maintain the two constants required to trigger rebuildings that result from deletions without referring to the inner state of the data structure, or the details of the old search trees' implementation. A rebuilding needs to be carried out following a deep insertion. Deep insertions can possibly be diagnosed by measuring insertion time. More realistically, the number of calls by the old search trees' layer to the layer under it can be counted. This number reflects the number number of nodes in the data structure that are accessed. The insertion of a deep node will cause the number of such calls to surpass a prespecified threshold.

Rebuildings as well as look-ups of weight unbalanced ancestors are carried out by a subroutine of the scapegoat layer. They modify the data only. Provided the data format is maintained, the old code shall work correctly with the rebuilt tree.

At the time of switch from the old code to the new code a single rebuilding of the whole tree is sufficient by Theorem 3.5.3. Thus the complexity of a switch is linear in the size of the data base.

The cost of "plugging in" scapegoat balancing can be amortized over the first $\Omega(size[T])$ update operations, ovoiding the rebuilding of the the whole tree required above. The scapegoat node can always be chosen at depth $\leq h_\alpha(n)$. Indeed, any node at depth $> h_\alpha(n)$ must have a weight unbalanced ancestor at depth $\leq h_\alpha(n)$ by Claim 3.5.1. If we always choose the scapegoat at depth $\leq h_\alpha(n)$ then we can prove:

Theorem 3.7.5 *The amortized complexity of* $\Omega(size[T])$ *update operations that use the* *scapegoat balancing scheme is* $O(\lg size[T])$ *starting with an arbitrary binary tree, for some scapegoat selection schemes.*

Proof: Notice that potential has to be maintained only at nodes that can be rebuilt. If we choose the scapegoat at depth $\leq h_\alpha(n)$, then we need to maintain potential at nodes of that depth only. Since

$$\sum_{\{x:d(x)=k\}} |size[left[x]] - size[right[x]]| \leq size[T]$$

the total potential that needs to be stored at an arbitrary binary tree to subsequently support scapegoat-balanced updates is $O(nh_\alpha(n)) = O(n \lg n)$. Amortizing this over $\Omega(n)$ operations gives the desired bound. □

Similarly, scapegoat balancing can be added onto the various tree-based schemes discussed in Section 3.7. This feature can also be used to support two-staged development of scapegoat code. The unbalanced structure produced in the first stage will then provide all of the complete system's features except performance.

3.8 Reducing Delete Incurred Restructuring

By rebuilding the whole tree whenever triggering condition (3.7)

$$size[T] > \alpha max\_size[T]$$

is satisfied the tree is guaranteed to stay loosely α-height-balanced (Theorem 3.5.3). That is

$$h(T) \leq h_\alpha(T) + 1. \tag{3.9}$$

We can reduce the frequency of Delete induced restructuring without violating the logarithmic depth of scapegoat trees. Call a binary tree **L-loosely** α-height-balanced if

$$h(T) \leq h_\alpha(T) + L. \tag{3.10}$$

In this section we show that if DELETE induced restructuring is triggered by the satisfaction of condition

$$size[T] > \alpha^L max\_size[T] \tag{3.11}$$

the tree can be kept L-loosely α-height-balanced.

Claim 3.8.1 *A scapegoat tree can handle a sequence of* INSERT *and* DELETE *operations with* DELETE *induced whole-tree restructuring being triggered by the satisfactions of*

$$size[T] > \alpha^L max\_size[T]$$

while remaining L-loosely α-height-balanced after each operation.

Proof: The proof is similar to that of Claim 3.5.10. Restricting our attention to the subsequence of operations $o_{i_1}, \ldots, o_{i_k}$ that modify $h_\alpha(T)$, we observe that by a generalization of Claim 3.5.9 an INSERT operation in this subsequence into an unbalanced tree reduces the looseness of the tree's balance by 1. Thus an INSERT operation in this subsequence that increases $max\_size[T]$ leaves the tree α-height-balanced. The degree of looseness of a tree is therefore upper bounded by the difference between the number of DELETE and INSERT operations in this subsequence since the last increase of $max\_size[T]$. Condition (3.11) guarantees that this quantity does not exceed L. □

Comment: The looser triggering condition for DELETE induced restructuring specified by (3.11) applies to multi-key scapegoat trees as well.

3.9 Comparison to Andersson's Work

We arrived at our result unaware of Andersson's publication [2] that has preceded our discovery by about a year. Even in light of his precedence scapegoat trees contribute to the theoretical understanding of the family of data structures that use partial rebuilding to enforce a bound on the tree's depth [3, 13, 55, 40, 25].

The first part of his thesis [3] culminates with the presentation of two data structures he calls $GB(c)$ trees and $GB_0(c)$ trees – General Balanced trees. These he terms "superclasses" containing all other classes of balanced trees. They satisfy the simplest possible criterion that guarantees logarithmic time searching – $O(\lg n)$ height. Most other tree based data structures, like AVL trees [1], Red-Black trees [9, 29], $BB(\alpha)$ trees [53], αBB trees [54] and Andersson's $BH(c)$ trees impose a balance condition that makes some of the trees of height $O(\lg n)$ not members of the class of allowed trees. Andersson's $GB(c)$ trees and $GB_0(c)$ trees are the first known scheme that legalizes all trees that can be searched in logarithmic time. The class of $GB_0(c)$ trees possesses the extra merit of storing no extra information at the nodes to support balancing. (Andersson's c is comparable to α by the equation $c = -(\lg \alpha)^{-1}$.)

Scapegoat trees likewise are a "superclass" of trees that does not maintain any balancing information at the nodes identical to $GB_0(c)$. A comparison of the main theorems for scapegoat trees and $GB_0(c)$ shows the greater generality of our theoretical analysis.

In particular Theorem 3.5.1 about rebuilding following an insertion is stronger than Andersson's comparable claim. After a deep node is inserted into a $GB_0(c)$ tree the path up to the root is retraced until the first node x satisfying

$$h(x) > \lceil c \lg size(x) \rceil$$

is encountered and rebuilt. He chooses the scapegoat using what we call the "alternative" method (equation (3.6)). Theorem 3.5.1 asserts that any weight unbalanced node on the path from the deep node to the root may be rebuilt to restore the balance. According to Claim 3.5.2 any node chosen by the "alternative" method is weight-unbalanced. Hence this method is only a special case of the more general analysis in Theorem 3.5.1.

As for deletion, Andersson proves that to handle the imbalances resulting from deletions it is sufficient to rebuild the whole tree whenever

$$d(T) \geq \gamma size[T],$$

where $d(T)$ is the number of deletions that were performed since the last rebuilding of the whole tree. He proves that $GB_0(c)$ trees compromise perfect height balance for $c \lg(1+\gamma)$-

loose height balance to accommodate deletions. That is nodes might be $c\lg(1+\gamma)$ levels deeper than the desired maximal depth of $h_{2^{-1/c}}(size[T])$.

We prove that to maintain a L-loosely height-balanced scapegoat tree it is sufficient to rebuild the whole tree whenever (equation (3.11)):

$$size[T] > \alpha^L max\_size[T]$$

or

$$max\_size[T] > \alpha^{-L} size[T].$$

Solving $2^{-1/c} = \alpha$, $c\lg(1+\gamma) = L$ we get

$$\gamma = \alpha^{-L} - 1.$$

Since

$$d(T) \geq max\_size[T] - size[T]$$

We conclude that our rebuilding criterion is theoretically stronger than Andersson's comparable criterion. To maintain the same balance condition we require less frequent rebuilding of the whole tree. Compare the number of DELETE-induced rebuildings of the whole tree in a scapegoat tree with parameter α to the number of such rebuildings for a $GB_0(c)$ with $c = -\lg^{-1}\alpha$.

Any sequence that causes the satisfaction of condition

$$max\_size[T] > \alpha^{-L} size[T] \tag{3.12}$$

must include at least $\gamma size[T] = (\alpha^{-L} - 1) size[T]$ deletions. Thus any sequence of operations that causes a DELETE-induced restructuring of a scapegoat tree will also trigger at least one restructuring of the comparable $GB_0(c)$ tree. The opposite is not true.

For example consider an arbitrarily long sequence of alternating INSERTs and DELETEs. For this input sequence a scapegoat tree does not get totally rebuilt even once, while a $GB_0(c)$ tree gets rebuilt arbitrarily many times.

To sum up a scapegoat trees require less total rebuilding to assure the same balance criterion as $GB_0(c)$ trees.

The usage of these schemes for multiway trees as well as their use for upgrading existing code are novel. Sleator and Tarjan [64] as well as Andersson [3] do not report experimental measurements of their suggested structures' performance. In the next section we present a practical study of tree-based dictionary solutions that do not require storage of balance enforcing information at the nodes.

3.10 Experimental Results

Our experiments address the efficient practical implementation of scapegoat trees and compare them to other known binary search trees' balancing schemes.

3.10.1 Optimizing Scapegoat Trees

The non-recursive method of rebuilding subtrees described in section 3.6.2 proved to work faster than the method described in section 3.6.1 by 25% – 30%. In section 3.4 we described two ways to choose the scapegoat. Our experiments suggest that checking for condition (3.6) yields a better overall performance.

In our experiments we used a variant of the non-recursive rebuilding algorithm described by the pseudo-code in section 3.6.2 which inserts all the nodes at the deepest level of the newly-built subtree at the leftmost possible positions, instead of spreading them evenly. This simplified the code somewhat and yielded a 6% – 9% percent speedup over the version described by the pseudo-code. Stout and Warren [68] call these route balanced trees. This issue is discussed further in Section 3.11.

It is natural to expect that the optimal value for parameter α should depend on the ratio between the number of searches and the number of modifications in a given sequence of requests to the scapegoat tree. The bigger the ratio of searches the more justified it is to reduce the value of α thereby enforcing a shallow tree even at the cost of more frequent rebuilding.

r \ N	1K	8K	64K
1	0.65	0.6	0.6
4	0.6	0.6	0.6
16	0.6	0.6	0.55
64	0.55	0.55	0.55
256	0.55	0.55	0.55
1024	0.55	0.55	0.55

Figure 3-4: The value of α for which scapegoat trees performed best as a function of N and r.

In table 3.10.1 we found experimentally the optimal value for α for different values of r and N, In some practical applications, both r and N or at least one of them might be predictable in advance at the time of implementation. In such cases we suggest using the results in table 3.10.1 to tune α.

3.10.2 Scapegoat Trees vs. Other Schemes

We compared scapegoat trees to two other schemes for maintaining binary search trees – red-black trees and splay trees. We also compare the performance of scapegoat trees for different values of α. We compare the performance for each one of the three operations Insert, Delete, and Search separately. We consider two types of workloads – uniformly distributed inputs and sorted inputs. The results are summarized in Tables 3.10.2 and 3.10.2. The tables list average time in seconds per $128K$ $(131,072)$ operations.

To compare the performance for uniformly distributed inputs, we inserted the nodes into a tree in a random order, then searched for randomly chosen nodes in the tree, and finally deleted all of the nodes in random order. We tried trees of three sizes – $1K$, $8K$ and $64K$. The results appear in Table 3.10.2.

Table 3.10.2 summarizes the results of the comparison for sorted sequences. Here too

		1 K			8 K			64 K		
		search	insert	delete	search	insert	delete	search	insert	delete
splay trees		2.65	5.20	6.05	3.90	6.72	7.76	5.38	8.15	9.18
red-black trees		6.28	7.13	8.54	8.95	10.10	11.41	12.19	13.42	14.87
scapegoat trees	α=0.55	1.65	9.46	3.49	2.28	10.14	4.13	3.16	9.61	5.13
	α=0.6	1.66	5.88	3.87	2.37	7.05	4.42	3.45	7.89	5.24
	α=0.65	1.73	4.93	4.49	2.49	6.45	4.81	3.51	7.82	5.56
	α=0.7	1.87	4.22	5.62	2.62	5.72	5.91	3.53	6.93	6.52
	α=0.75	1.87	3.83	6.88	2.71	4.70	7.73	3.73	5.98	8.84

Figure 3-5: Results of comparative experiments for uniformly distributed inputs. Execution time in seconds per $128K$ $(131,072)$ operations for splay trees, red-black trees and scapegoat trees with α varying between $0.55 - 0.75$ for tree sizes of $1K$, $8K$ and $64K$.

we tried three tree sizes – $1K$, $8K$ and $64K$. First we inserted the nodes into a tree in increasing order of keys, then we searched for all of the keys that were inserted in increasing order, and finally we deleted all of the nodes in increasing order of keys.

For uniformly distributed sequences our experiments show that one can choose an α so that scapegoat trees outperform red-black trees and splay trees on all three operations. However, for the insertion of sorted sequences scapegoat trees are noticeably slower than the other two data structures. Hence, in practical applications, it would be advisable to use scapegoat trees when the inserted keys are expected to be roughly randomly distributed, or when the application is search intensive.

For the splay trees we used top-down splaying as suggested by Sleator and Tarjan [64]. The implementation of red-black trees follows Chapter 14 in Cormen, Leiserson and Rivest [20].

3.11 Discussion and Conclusions

Stout and Warren [68] present an algorithm which takes an arbitrary binary search tree and rebalances it to form what they call a *route balanced tree* using linear time and only constant space. This improves upon the logarithmic space required to output a perfectly balanced tree. A route balanced tree is one containing exactly 2^d nodes at level d for $1 \leq g < \lfloor \lg n \rfloor$, with no limitation on the position of trees at the deepest level. Can the

		1 K			8 K			64 K		
		search	insert	delete	search	insert	delete	search	insert	delete
splay trees		2.83	7.75	6.17	3.59	9.49	7.13	4.96	11.19	8.49
red-black trees		2.40	3.38	3.50	2.65	3.21	3.81	2.68	3.46	3.74
scapegoat trees	α=0.55	1.37	21.25	2.90	1.88	29.84	3.45	2.50	36.91	3.72
	α=0.6	1.41	18.67	3.08	1.89	25.38	3.31	2.44	33.26	3.75
	α=0.65	1.38	16.17	3.04	1.88	23.39	3.19	2.58	29.47	3.93
	α=0.7	1.36	15.80	3.02	1.90	20.02	3.36	2.59	24.31	3.87
	α=0.75	1.54	13.94	3.83	1.88	19.25	3.49	2.67	24.79	4.15

Figure 3-6: Results of comparative experiments for monotone inputs. Execution time in seconds per $128K$ $(131,072)$ operations for splay trees, red-black trees and scapegoat trees with α varying between $0.55 - 0.75$ for tree sizes of $1K$, $8K$ and $64K$.

performance of scapegoat trees be achieved by a structure resorting to route rebalancing rather than perfect rebalancing of subtrees?

We also leave as an open problem the average-case analysis of scapegoat trees (say, assuming that all permutations of the input keys are equally likely).

Section 3.4.2 proposed a few ways in which the scapegoat node can be chosen. Which one is superior remains an open question that may be resolved theoretically or experimentally.

To summarize: scapegoat trees are the first "unencumbered" tree structure (i.e., having no extra storage per tree node) that achieves a worst-case SEARCH time of $O(\log n)$, with reasonable amortized update costs.

Acknowledgments

We are greatful to Jon Bentley for suggesting the applicability of our techniques to $k-d$ trees. We thank Charles Leiserson for some helpful discussions. We also thank David Williamson and John Leo for allowing us to use their software in our experiments.

Bibliography

[1] G. M. Adel'son-Vel'skiĭ and E. M. Landis. An algorithm for the organization of information. *Soviet Mathematics Doklady*, 3:1259–1263, 1962.

[2] A. Andersson. Improving partial rebuilding by using simple balance criteria. In *Proceedings of the Workshop on Algorithms and Data Structures*, pages 393–402. Springer-Verlag, 1989.

[3] A. Andersson. *Efficient Search Trees.* PhD thesis, Department of Computer Science, Lund University, Sweden, January 1990.

[4] D. Angluin. Queries and concept learning. *Machine Learning*, 2:319–342, 1988.

[5] J. Aslam. *Noise Tolerant Algorithms for Learning and Searching.* PhD thesis, MIT, February 1995. MIT/LCS/TR-657.

[6] J. Aslam and A. Dhagat. Searching in the presence of linearly bounded errors. In *Proceedings of the Twenty Third Annual ACM Symposium on Theory of Computing*, pages 486–493. ACM Press, 1991.

[7] J. L. Baer and B. Schwab. A comparison of tree balancing algorithms. *Communications of the ACM*, 20(5):322–330, May 1977.

[8] J.M. Barzdin and R.V. Freivalds. On the prediction of general recursive functions. *Soviet. Math. Dokl.*, 13:1224–1228, 1972.

[9] R. Bayer. Symmetric binary B-trees: Data structure and maintenance algorithms. *Acta Informatica*, 1:290–306, 1972.

[10] S. Ben-David and E. Dichterman. Learning with restricted focus of attention. In *Proceedings of the Sixth Annual Workshop on Computational Learning Theory*, pages 287–296. ACM Press, 1993.

[11] S. Ben-David and E. Dichterman. Learnability with restricted focus of attention guarantees noise-tolerance. In *Fifth International Workshop on Algorithmic Learning Theory*, pages 248–259. Springer-Verlag, 1994.

[12] Jon L. Bentley. Multidimensional binary search trees used from associative searching. *Communications of the ACM*, 19:509–517, 1975.

[13] Jon L. Bentley. Multidimensional binary search trees in database applications. *IEEE Transactions on Software Engineering*, 5(4):333–340, 1979.

[14] A. Birkendorf, E. Dichterman, J. Jackson, N. Klasner, and H. U. Simon. On restricted-focus-of-attention learnability of boolean functions. In *Proceedings of the Ninth Annual Workshop on Computational Learning Theory*, pages 205–216. ACM Press, 1996.

[15] A. Blum, L. Hellerstein, and N. Littlestone. Learning in the presence of finitely or infinitely many irrelevant attributes. In *Proceedings of the Fourth Annual Workshop on Computational Learning Theory*, pages 157–166. Morgan Kaufmann, August 1991.

[16] Avrim Blum. Learning boolean functions in an infinite attribute space. In *Proceedings of the Twenty Second Annual ACM Symposium on Theory of Computing*, pages 64–72, Baltimore, Maryland, May 1990. ACM Press.

[17] N. Cesa-Bianchi, Y. Freund, D. Helmbold, D. Haussler, R. Schapire, and M. Warmuth. How to use expert advice. In *Proceedings of the Twenty Fifth Annual ACM Symposium on Theory of Computing*, pages 382–391, San Diego, CA, May 1993. ACM Press.

[18] N. Cesa-Bianchi, Y. Freund, D.P. Helmbold, and M. Warmuth. On-line prediction and conversion strategies. In *EuroCOLT*, pages 205–216. Clarendon Press, 1993.

[19] V. Chvátal. A greedy heuristic for the set-covering problem. *Mathematics of Operations Research*, 4(3):233–235, 1979.

[20] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. MIT Press/McGraw-Hill, 1990.

[21] G. Cornuejols, M.L. Fisher, and G.L. Nemhauser. Location of bank accounts to optimize float: An analytical study of exact and approximate algorithms. *Management Science*, 23:789–810, 1977.

[22] J. Czyzowicz, D. Mundici, and A. Pelc. Ulam's searching game with lies. *Journal of Combinatorial Theory, Ser. A*, 52:62–76, 1989.

[23] George B. Dantzig. *Activity Analysis of Production and Allocation*, chapter Programming of Interdependent Activities, II, Mathematical Models, pages 19–32. John Wiley and Sons Inc., New York, 1951.

[24] U. Feige, D. Peleg, P. Raghavan, and Upfal E. Computing with unreliable information. In *Proceedings of the Twenty Second Annual ACM Symposium on Theory of Computing*, pages 128–137. ACM Press, 1990.

[25] R. A. Finkel and J. L. Bentley. Quad-trees; a data structure for retrieval on composite keys. *Acta Informatica*, 4:1–9, 1974.

[26] M.L. Fisher and D.S. Hochbaum. Probabilistic analysis of the planar k-median problem. *Mathematics of Operations Research*, 5(1):265–294, Feb 1980.

[27] I. Galperin. Analysis of greedy expert hiring and an application to memory-based learning. In *Proceedings of the Ninth Annual Conference on Computational Learning Theory*, pages 217–223. ACM Press, 1996.

[28] I. Galperin and R. Rivest. Scapegoat trees. In *Proceedings of the 4th ACM-SIAM Symposium on Discrete Algorithms*, pages 165–174. ACM SIAM, 1993.

[29] Leo J. Guibas and Robert Sedgewick. A diochromatic framework for balanced trees. In *Proceedings of the 19th Annual Symposium on Foundations of Computer Science*, pages 8–21. IEEE Computer Society, 1978.

[30] W. Guzicki. Ulam's searching game with two lies. *Journal of Combinatorial Theory, Ser. A*, 54:1–19, 1990.

[31] Chang H. and Iyengar S. S. Efficient algorithms to globally balance a binary search tree. *Communications of the ACM*, 27(7):695–702, July 1984.

[32] D. Haussler, J. Kivinen, and M.K. Warmuth. Tight worst-case loss bounds for predicting with expert advice. In *EuroCOLT*, pages 205–216. Clarendon Press, 1995.

[33] D. Haussler and P. Long. A generalization of Sauer's lemma. Technical Report UCSC-CRL-90-15, U.C. Santa Cruz Computer Research Laboratory, Apr 1990.

[34] David Haussler. Generalizing the PAC model for neural net and other learning applications. Technical Report UCSC-CRL-89-30, University of California Santa Cruz, Sep 1989.

[35] David Haussler. Decision theoretic generalizations of the PAC model for neural net and other learning applications. *Information and Computation*, 100(1):78–150, Sep 1992.

[36] D. S. Johnson. Approximation algorithms for combinatorial problems. *Journal of Computer and System Sciences*, 9:256–278, 1974.

[37] N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4(4):373–395, 1984.

[38] Michael J. Kearns and Umesh V. Vazirani. *An Introduction to Computational Learning Theory.* The MIT Press, 1994.

[39] L.G. Khachiyan. A polynomial algorithm in linear programming. *Soviet Mathematics Doklady*, 20:191–194, 1979.

[40] George S. Leuker. A data structure for orthogonal range queries. In *Proceedings of the 19th Annual Symposium on Foundations of Computer Science*, pages 28–34. IEEE Computer Society, 1978.

[41] Jyh-Han Lin and Jeffrey Scott Vitter. ϵ-approximation with minimum packing constraint violation. In *Proceedings of the Twenty Fourth Annual ACM Symposium on Theory of Computing*, pages 771–782. ACM Press, May 1992.

[42] Jyh-Han Lin and Jeffrey Scott Vitter. A theory for memory-based learning. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, pages 103–115. ACM Press, Jul 1992.

[43] Nick Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2:285–318, 1988.

[44] Nick Littlestone and Manfred K. Warmuth. The weighted majority algorithm. *Information and Computation*, 108(2):212–261, February 1994.

[45] L.H. Loomis and S. Sternberg. *Advanced Calculus*. Addison-Wesley, 1968.

[46] C. Lund and M. Yanakakis. On the hardness of approximating minimization problems. In *Proceedings of the Twenty Fifth Annual ACM Symposium on Theory of Computing*, pages 286–293. ACM Press, 1993.

[47] D. Marr. A theory for cerebral neocortex. In *Proceedings of the Royal Society of London B.*, 176, pages 161–234, 1970.

[48] N. Megiddo and K.J. Supowit. On the complexity of some common geometric location problems. *SIAM Journal on Computing*, 13:182–196, 1984.

[49] K. Mehlhorn and A. Tsakalidis. Data structures. In J. van Leeuwen, editor, *Algorithms and Complexity*, volume A, chapter 6, pages 301–341. Elsevier, 1990.

[50] G.L. Nemhauser and L.A. Wolsey. Best algorithms for approximating the maximum of a submodular set function. *Mathematics of Operations Research*, 3(3):177–188, 1978.

[51] G.L. Nemhauser and L.A. Wolsey. Maximizing submodular set functions: Formulations and analysis of algorithms. In P. Hansen, editor, *Studies on Graphs and Discrete Programming*, volume 11 of *Annals of Discrete Mathematics*, pages 279–301. North-Holland, 1981.

[52] G.L. Nemhauser, L.A. Wolsey, and M.L. Fisher. An analysis of approximations for maximizing submodular set functions-I. *Mathematical Programming*, 14:265–294, 1978.

[53] J. Nievergelt and E. M. Reingold. Binary trees of bounded balance. *SIAM Journal on Computing*, 2(1):33–43, 1973.

[54] H. J. Olivie. A new class of binary search trees: Half balanced binary search trees. *International J. of Computer Mathematics*, 9:287–303, 1981.

[55] Mark H. Overmars and Jan van Leeuwen. Dynamic multi-dimensional data structures based on quad- and $k-d$ trees. *Acta Informatica*, 17:267–285, 1982.

[56] Guillermo Owen. *Game Theory*. Academic Press, 1982.

[57] C.H. Papadimitriou. Worst-case and probabilistic analysis of a geometric location problem. *SIAM Journal on Computing*, 10:542–557, 1981.

[58] K.R. Parthasarathy. *Introduction to Probability and Measure*. Springer-Verlag, 1977.

[59] A. Pelc. Solution of Ulam's problem on searching with a lie. *Journal of Combinatorial Theory, Seires A*, 44:129–140, 1987.

[60] A. Pelc. Searching with known error probability. *Theoretical Computer Science*, 63:185–202, 1989.

[61] R.L. Rivest, A. R. Meyer, D.J. Kleitman, Winklmann K., and J. Spencer. Coping with errors in binary search procedures. *Journal of Computer and System Sciences*, 20:396–404, 1980.

[62] Hanan Samet. Deletion in two-dimensional quad trees. *Communications of the ACM*, 23(12):703–710, 1980.

[63] Lloyd S. Shapley. Cores of convex games. *International Journal of Game Theory*, 1(1):11–26, 1971.

[64] Daniel D. Sleator and Robert E. Tarjan. Self-adjusting binary search trees. *Journal of the ACM*, 32(3):652–686, 1985.

[65] D.D. Sleator and R.E. Tarjan. Amortized efficiency of list update rules. In *Proceedings of the Sixteenth Annual ACM Symposium on Theory of Computing*, pages 488–492. ACM Press, 1984.

[66] J. Spencer. Ulam's searching game with a fixed number of lies. *Theoretical Computer Science*, 95:307–322, 1992.

[67] J. Spencer and P. Winkler. Three thresholds for a liar. *Combinatorics, Probability and Computing*, 1:81–93, 1992.

[68] Q. F. Stout and B. L. Warren. Tree rebalancing in optimal time and space. *Communications of the ACM*, 29(9):902–908, September 1986.

[69] S. Ulam. *Adventures of a Mathematician.* Scribners (New York), 1977.

[70] V.G. Vovk. Agregating strategies. In *Proceedings of the Third Annual Workshop on Computational Learning Theory*, pages 371–383. Morgan Kaufmann Publishers, 1990.