MIT/LCS/TR-348

MIT/LCS/TR-348

ON PLAYING WELL IN A SUM OF GAMES

YEDWAB

Laura Yedwab

ON PLAYING WELL IN A SUM OF GAMES

This blank page was inserted to preserve pagination.

On Playing Well in a Sum of Games

by

Laura Jo Yedwab

B.S.E., University of Pennsylvania
(1980)

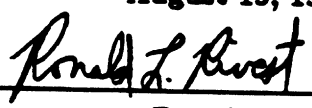
Submitted in partial fulfillment of
the requirements for the degree of

Master of Science
in
Electrical Engineering and Computer Science

at the
Massachusetts Institute of Technology
August, 1985

© Laura Jo Yedwab 1985
The author hereby grants to MIT permission to reproduce and
to distribute copies of this thesis document in whole or in part.

Signature of Author
Department of Electrical Engineering and Computer Science
August 19, 1985

Certified by 
Ronald Rivest
Professor of Electrical Engineering and Computer Science

Accepted by _____
Arthur Smith
Chairman, Department Committee on Graduate Students

On Playing Well in a Sum of Games [1]

by

Laura Jo Yedwab

Submitted to the
Department of Electrical Engineering and Computer Science
on August 22, 1985
in partial fulfillment of the requirements for the degree of
Master of Science
in
Electrical Engineering and Computer Science

Abstract

Many games are naturally described as a *sum* of games, *e.g.*, nim and the endgame of Go. Let $G_1, \ldots, G_n$ represent n games. Then a move in the sum $G_1 + \cdots + G_n$ consists of picking a component game G_i and making a move in G_i. This thesis analyzes play in a sum of games from three different perspectives: computational complexity, approximate solutions, and optimal search algorithms.

Lockwood Morris[17] proves that the problem of determining the optimal strategy in a sum of games is $PSPACE$-complete. This thesis proves that the problem is $PSPACE$-complete even when the component games are so simple that they can be represented as depth two trees.

Hanner [7] shows that the value of a sum of games can approximated to within the maximum *temperature* of the component games. This thesis presents a clear and concise proof of Hanner's bounds. This thesis also improves upon Hanner's result. It shows that the value of a sum of games can be approximated to within the *second* highest temperature.

This thesis describes how Berliner's B^* search algorithm[4] can be effectively combined with the approximate solutions to speed up the search for an optimal solution.

Thesis Supervisor: Dr. Ronald Rivest
Title: Professor of Electrical Engineering and Computer Science

[1] This research was supported in part by a AT&T graduate fellowship

Acknowledgments

I would like to thank my advisor, Ron Rivest, for his support during this research. His guidance and insight were invaluable.

Joe Buhler brought Morris's $PSPACE$-completeness proof to my attention. Joe Buhler and Jean Michel read early drafts of the thesis, and provided useful comments.

Ray Hirschfeld's constant willingness to help went beyond the call of duty.

I would like to thank John Conway, Philip Klein, Charles Leiserson, Peter Shor, and Daniel Weise for helpful discussions concerning this work and its presentation.

Finally, I gratefully thank AT&T for providing me with a graduate fellowship.

Contents

1 Introduction **1**

2 Basics **4**
- 2.1 Games 4
- 2.2 Sum of Games 7
- 2.3 Sente 7
- 2.4 The Negative of a Game 9
- 2.5 Mean Value 10
- 2.6 Taxation 11
  - 2.6.1 Temperature 12
  - 2.6.2 The Value of a Taxed Game 13
- 2.7 Thermographs 15
- 2.8 Mean Value Revisited 18

3 Easy Games - Hard Games **19**
- 3.1 Switches 19
- 3.2 Stacks of Coins 20
- 3.3 Left Heavy Games 21
  - 3.3.1 C-Left Heavy Games 21
  - 3.3.2 Log(n)-Left Heavy Games 25
  - 3.3.3 n-Left Heavy Games 26

4 Complexity of SUM **29**
- 4.1 $SUM_{d\leq 2}$ is NP-hard 30
- 4.2 $SUM_{d\leq 2}$ is $PSPACE$-complete 37

5 Coping with SUM 46
5.1 Heuristic Solutions 47
5.1.1 Follow the Leader Strategy 48
5.1.2 Mean Strategy 49
5.1.3 Improving Hanner's Bounds 59
5.1.4 Thermostatic Strategy 63
5.1.5 Comparing Results 65
5.2 Search 66
5.2.1 The B^* Algorithm 67

6 Endgame of Go 72
6.1 Ko 74
6.2 Complexity Results 76
6.3 Approximation and Search Algorithms 79

7 Future Research 80
7.1 Complexity 80
7.2 Heuristic Solutions 82
7.3 Search 83

List of Figures

1.1 A *Dots and Boxes* Game as a Sum of Games 2

2.1 Defining H_1 5
2.2 $V_L(H_2), V_R(H_2), L_n(H_2)$ and $R_n(H_2)$ 6
2.3 Optimal Play in a Sum of Two Games 8
2.4 Defining H_3 and H_4 9
2.5 Defining H_5, H_6, and H_7 10
2.6 Defining H_8 12
2.7 Defining H_9 and H_{10} 13
2.8 Computing V_L and V_R for H_{11} and H_{12} 15
2.9 Thermograph for $\{ 5 \mid -5 \}$ 16
2.10 Defining H_{13} 17
2.11 Constructing the Thermograph for $\{\{ 5 \mid -5 \} \mid 20 \}$ 17
2.12 Constructing the Thermograph for H_{13} 17

3.1 Stacked Coins 20
3.2 Representing a Stack of Coins 21
3.3 A Left Heavy Tree where Left's Move is *Sente* 22
3.4 A 1-Left Heavy Game 25
3.5 Defining H_{15}, H_{16}, and H_{17} 27
3.6 The Importance of Context 28

4.1 Constructing an Instance of ALT 32
4.2 Comparing the Structure of the Proofs 38
4.3 Transforming a Formula into an Instance of 2P-EXACT-COVER 40

5.1 Comparing the Accuracy of Hanner's and Milnor's Results . 50

5.2 t-optimal moves in H_{22} 52
5.3 A Sum Where the First Move Is Not Sente 53
5.4 A Sum Where the Locally Optimal Response Is Incorrect . 53
5.5 Hanner's Bounds as G' Is Played 60
5.6 Constructing the Thermograph for H' 61
5.7 Defining H_{23}, H_{24}, and H_{25} 63
5.8 Defining H_{26} and H_{27} 65
5.9 Defining H_{28} and H_{29} 66
5.10 Defining H_{30}, H_{31}, and H_{32} 67
5.11 Defining H_{33}, H_{34}, and H_{35} 68
5.12 Expanding the Search Tree One Level 69
5.13 The Search Tree Expanded using *Disprove-Rest* 70
5.14 The Search Tree Expanded using *Prove Best* 71

6.1 An end position in a game of Go 73
6.2 *Chōsei* 74
6.3 Ko 75
6.4 Representing a Left Heavy Tree 77
6.5 A Key Game in the $PSPACE$-complete Proof 77

7.1 A Sum of All Small Games 81

Chapter 1

Introduction

Let $G_1, \ldots, G_n$ represent n games. The sum,

$$G_1 + \cdots + G_n,$$

is a game in which a move consists of picking a component game G_i and making a move in G_i. The sum is over when a player is unable to make a move in any of the component games.

Many games are naturally described as a sum of games. Nim is the sum of a number of simple heap games. A hackenbush game is a sum of a number of distinct hackenbush pictures.

Other common examples occur when, during the course of a game, a single integrated game becomes decomposed into a sum of games. For example, when the game *dots and boxes* is played, the board quickly becomes divided up into a number of isolated *dots and boxes* games. Figure 1.1 shows a *dots and boxes* game that is the sum of three distinct games. To move in the sum, a player decides which area to play in, and then draws a line in that area.

The game of Go is another example. Towards the end of the game, the black and white stones divide the board into separate territories. The endgame of Go revolves around a number of small, independent, border disputes and is naturally characterized as a sum of games. To move in the sum, a player decides with area to play in, and then places a stone in that area.

The endgame of Go is an important example because of the wide spread interest in the game. Go is played professionally in the Orient at the level

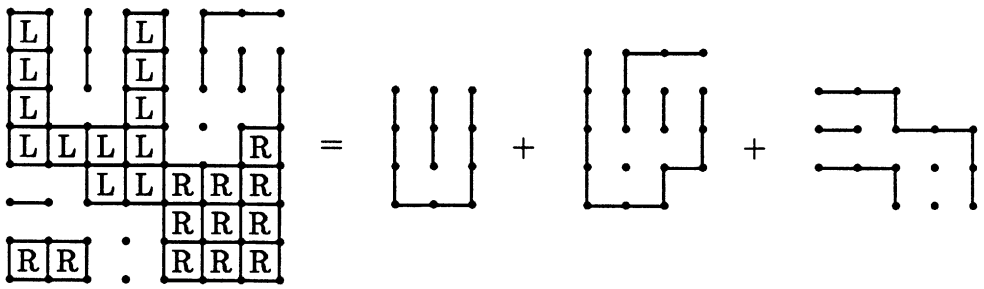


Figure 1.1: A *Dots and Boxes* Game as a Sum of Games

that chess is played in the West. Furthermore, Go has attracted a reasonable amount of attention from the A.I. community as being a good domain for studying problem solving issues, *e.g.*, [1], [13], [23], and [24]. In fact, the endgame of Go was the author's original motivation for studying sums of games.

This thesis analyzes the play in a sum of games from three different perspectives: computational complexity, approximate solutions, and optimal search algorithms.

Call SUM the problem of determining the winning strategy in a sum of games. The goal, in analyzing the computational complexity of SUM, is to determine the theoretical bound on how efficient any algorithm for solving SUM or a restricted subclass of SUM can be. Lockwood Morris[17] proves that SUM is $PSPACE$-complete. Hence, assuming that $P \neq PSPACE$, no polynomial time, optimal strategy exists for playing an arbitrary sum of games. However, Morris's proof leaves open the possibility that there exist interesting restricted subclasses of SUM that can be solved in polynomial time.

This thesis narrows the gap between those problems known to be in P and those that are known to be $PSPACE$-complete. It proves that SUM is $PSPACE$-complete even when the component games are so simple that they can be represented as depth two trees.

Since SUM is $PSPACE$-complete, it is unlikely that an efficient algorithm for solving it will be discovered. Two alternate approaches exist for coping with an instance of SUM. The first is to relax the criteria of success.

Instead of requiring an optimal solution, an algorithm is only required to produce a solution that is close to optimal. The second alternative is to do an exponential time search for the optimal solution.

Approximate solutions are beneficial when speed is critical and small errors in the solution are tolerable. Hanner [7] proves that the value of a sum of games can approximated to within the maximum *temperature* of the component games. These bounds are surprisingly accurate. They are independent of the number of component games in the sum and independent of the complexity of the component games. This thesis presents a clear and concise proof of Hanner's bounds.

This thesis also improves upon Hanner's result. It show that the value of a sum of games can be approximated to within the second highest temperature.

When the optimal solution to an instance of SUM is required, the only currently available technique is exponential time search. However, the approximate solution described in this thesis provide a great deal of power in directing and pruning search. In particular, this thesis describes how Berliner's B^* search algorithm[4] can be effectively combined with the approximate solutions to speed up the search for an optimal solution.

The remainder of this thesis is organized as follows: Chapter 2 gives the basic definitions used through out the thesis. This includes the definition of a game, a sum of games, and taxation. Chapter 3 analyzes optimal play for some simple restricted classes of SUM. Chapter 4 proves that SUM is $PSPACE$-complete even when the component games have depth two or less. Chapter 5 considers the two basic approaches for coping with a sum of games: approximate solutions and optimal search algorithms. Chapter 6 considers how the theory of sums of games as described in this thesis applies to the endgame of Go. Chapter 7 summarizes the results and considers future research directions.

Chapter 2

Basics

Conway developed a unified theory of numbers and games. The theory is beautiful, elegant, and fun. The reader is encouraged to read [5], [11], and [2] for an extensive introduction. [5] provides a presentation of the whole theory. [11] presents a simple introduction to the development of numbers. [2] presents an extensive treatment of the theory with respect to games.

This chapter presents the portion of the theory used in this thesis. In particular the notion of game, sums of games, and taxation is described. However, the chapter does not go into the development of numbers, but assumes that numbers, such as 5 and $-3\frac{1}{2}$, are well defined.

2.1 Games

Consider a two person, zero sum, perfect information game like chess, checkers, sprouts, or nim. Let the two players be called Left and Right. It is natural to represent such a game as a tree, where

- the nodes represent positions,
- the root is the initial position,
- an edge between node a and b represents a valid move from position a to position b,
- leaf nodes represent stopping positions, and
- the value of a leaf represents the payoff to Left when the game reaches that final position.

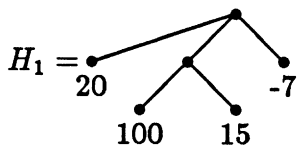
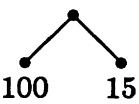


Figure 2.1: Defining H_1

By definition, Left wins the game if the final score (his payoff) is greater than zero, or if the the final score equals zero and its Right's turn to play when the game ends.

Moves for Left are distinguished from moves for Right by the direction of the edges. In particular, moves for Left are represented by edges that go down and to the left. Moves for Right are represented by edges that go down and to the right.

For example, consider the game shown in Figure 2.1. If Right plays first, then his only move his to -7. He collects 7 from Left. If Left plays first, then Left has two options. He can either move to 20, thus ending the game and collecting 20 from Right, or he can move to:

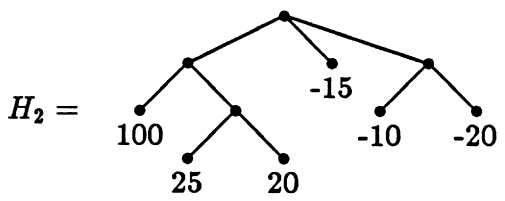


and when Right responds, Left collects 15. Since Left prefers gaining 20 over gaining 15, Left's optimal move is to 20.

This representation of a game is slightly unconventional in that at every position moves for both Left and Right are represented. Representing moves for both players at every position is necessary to handle sums of games.

The following notation is used to specify a game in a more textual way than as a picture of a tree: if $L_1, \ldots, L_n$ and $R_1, \ldots, R_n$ are games then

$$\{ L_1, \cdots, L_n \mid R_1, \cdots, R_n \}$$



$V_L(H_2) = 25$ $\qquad$ $V_R(H_2) = -15$
$L_1(H_2) = \{\, 100 \mid \{\, 25 \mid 20 \,\} \,\}$ $\qquad$ $R_1(H_2) = -15$
$L_2(H_2) = \{\, 25 \mid 20 \,\}$
$L_3(H_2) = 25$

Figure 2.2: $V_L(H_2), V_R(H_2), L_n(H_2)$ and $R_n(H_2)$

is a game where Left has the option of moving to one of the L_i games, and Right has the option of moving to one of the R_i games. For example, in this notation, the game shown in Figure 2.1 becomes:

$$\{\, 20, \{\, 100 \mid 15 \,\} \mid -7 \,\}$$

The following notation is used to specify the position a game reaches after n moves and the value of a game:

$L_n(G) \Rightarrow$ The position that results after n moves when both players play alternately, both players play optimally, and Left starts.

$R_n(G) \Rightarrow$ has the same meaning as $L_n(G)$ except that Right starts the play in G.

$V_L(G) \Rightarrow$ is the final score when both players play alternately, both players play optimally, and Left starts.

$V_R(G) \Rightarrow$ is the same as $V_L(G)$ except that Right starts the play in G.

For example, Figure 2.2 shows the notation applied to H_2.

This thesis only considers games for which both players want to play, *i.e.*, a player gains more by playing in the game than by allowing his opponent to play. More precisely, for all games G and for each position G' in G:

$$V_L(G') \geq V_R(G').$$

When the condition does not hold, the game is called a number. Again the reader is referred to [5] and [11] for a more detailed treatment of numbers.

2.2 Sum of Games

Let $G_1, \ldots, G_n$ represent n games. Then a move in the sum

$$G = G_1 + \cdots + G_n$$

consists of picking a component game G_i and making a move in G_i. G terminates when each component game, G_i, is reduced to a number. The final score is the sum of the final positions.

For example, Figure 2.3 shows optimal play in the sum of two games. Left begins by playing in the second component game and thus preventing Right from gaining -80. Right is not obliged to respond in the same game that Left played in. In fact, Right's optimal move is to play in the first component game. Finally, on the third move, Left plays in the only remaining game. The final score is positive, so Left wins.

As with a single game, Left's optimal strategy in a sum of games is to maximize the final score. However, unlike a single game where the optimal strategy can be determine efficiently by a min-max procedure, there appears to be no fast algorithm for determining the optimal strategy in a sum of games.

2.3 Sente

Sente is a Japanese word often used to describe a move in a Go game. A move in a sum of games is *sente* if it forces the opponent to respond locally, *i.e.*, in the same component game.

The notion of *sente* is important in understanding optimal play in a sum of games. For example, consider H_3, and H_4 shown in Figure 2.4. Left's

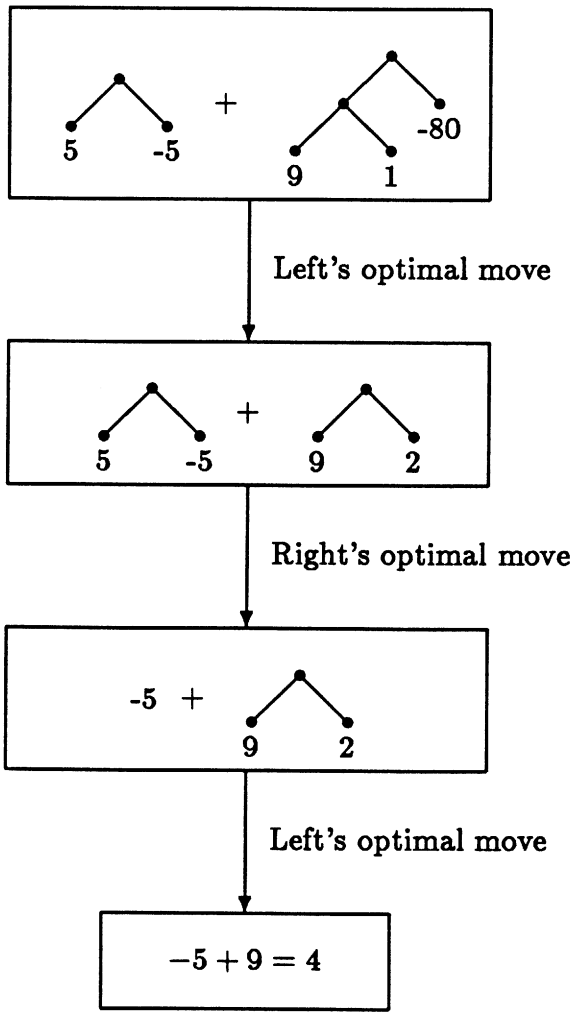


Figure 2.3: Optimal Play in a Sum of Two Games

Figure 2.4: Defining H_3 and H_4

optimal move in game H_3 is to collect 15. Left's optimal move in H_4 is to collect -10. However, if Left plays either of these locally optimal moves in $H_3 + H_4$, then Right will respond in the other component game and win. Left's optimal move in $H_3 + H_4$ is to $\{\, 100 \mid 12 \,\}$. This forces Right to respond locally and take 12. Left is then able to play first in the second component game. The final score is $12 - 10 = 2$ and Left wins.

Left's move to $\{\, 100 \mid 12 \,\}$, though not locally optimal, forces a response from Right. It is a *sente* move. Using chess terminology, it allows Left to keep tempo.

2.4 The Negative of a Game

The negative of a game is obtained by reversing the roles of Left and Right. In particular, if $G = \{\, L \mid R \,\}$, then the negative of G is defined as follows:

$$-G = \{\, -R \mid -L \,\}$$

For example:

$$\begin{aligned} -\{\, 5 \mid 4 \,\} &= \{\, -4 \mid -5 \,\} \\ -\{\, \{\, 10 \mid 5 \,\} \mid -4 \,\} &= \{\, 4 \mid \{\, -5 \mid -10 \,\} \,\} \end{aligned}$$

For any game G, the first player to move in

$$G + (-G)$$

will always lose. Without loss of generality, assume Left starts. For every move Left can make, Right can respond in the other component game with

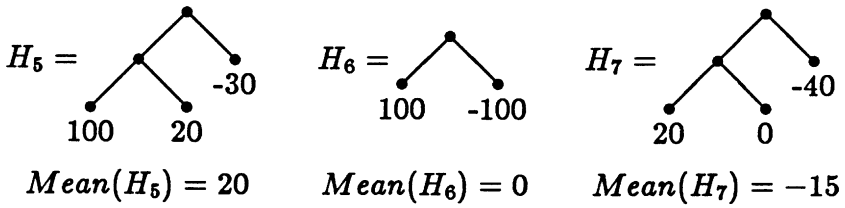


Figure 2.5: Defining H_5, H_6, and H_7

the negative of Left's move. Hence, Right can cancel any gain Left makes by going first. The final score will be zero with Left to play, but he has no play, so he loses.

2.5 Mean Value

The mean value of a game was first defined by Hanner[7] and used to approximate the value of a sum of games. The mean value of a game measures the inherent worth of a game not counting the advantage a player gains by moving first.

Let nG represent the sum of n copies of G, *i.e.*,

$$\overbrace{G + \cdots + G}^{n}.$$

The mean value of G is defined to be

$$Mean(G) = \lim_{n \to \infty} \frac{V_L(nG)}{n}$$

The limit always exists [7].

One way to compute the mean value of G is to analyze optimal play in nG for large n. This is best described through examples. What follows is a description of how the mean value for each of the three games shown in Figure 2.5 is computed.

Consider optimal play in the game nH_5. Every time Left plays, Left reduces a component game to $\{ 100 \mid 20 \}$. In doing so, Left threatens to

take 100. This forces Right to respond and reduce the switch to 20. Hence, each component game of nH_5 is reduced to 20, $V_L(nH_5) = 20n$, and

$$Mean(H_5) = \lim_{n\to\infty} \frac{n * 20}{n} = 20$$

Next, consider optimal play in the game nH_6 where n is even. Since the mean value is defined as a limit as n goes to infinity, no generality is lost by assuming that n is even. Every play by Left in nH_6 is balanced by a play by Right, *i.e.*, every time Left plays in a switch and takes 100, Right responds in another switch and takes -100. Hence, $V_L(nH_6) = 0$ and

$$Mean(H_6) = \lim_{n\to\infty} \frac{0}{n} = 0$$

Finally, consider play in the game nH_7. Without loss of generality, assume that n is a multiple of 4. Play is divided into two stages. During the first stage, Left's optimal move is always to reduce one of the H_7 games to $\{\, 20 \mid 0 \,\}$ and Right's optimal response is to reduce one of the H_7 games to -40. At the end of the first stage, the game has been reduced to:

$$\overbrace{\{\, 20 \mid 0 \,\} + \cdots + \{\, 20 \mid 0 \,\}}^{\frac{n}{2}} + \frac{-40n}{2}.$$

During the second stage, Left and Right will play out the $\frac{n}{2}$ switches. The final result is that half of the H_7 games are reduced to -40, a quarter of them are reduced to 20 and a quarter of them are reduced to 0. Therefore:

$$Mean(H_7) = \lim_{n\to\infty} \frac{\frac{-40n}{2} + \frac{20n}{4} + \frac{0n}{4}}{n} = \frac{-60}{4} = -15$$

Determining the mean value of a game G by analyzing play in nG is cumbersome. Section 2.8 provides an efficient method for computing the mean value of a game.

2.6 Taxation

It is useful to consider the value of a game when a tax of t is imposed on every move in the game. When a game is taxed and its Left's turn to play,

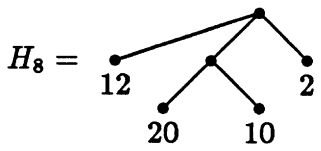


Figure 2.6: Defining H_8

Left either pays Right t and makes a move, or passes. Similarly, on Right's turn, Right either pays Left t and makes a move, or Right passes.

For example, consider play in H_8 (Figure 2.6) when the tax is 3. If Left pays 3 to take 12, then he gains 9. However, if Left pays 3 to move to $\{ 20 \mid 10 \}$, then when Right responds, Right pays 3 to move to 10. The taxes Left paid to Right and the taxes Right paid to Left balance out. Left gets a net gain of 10. Hence, when $t = 3$, Left's optimal move is to $\{ 20 \mid 10 \}$.

Though it is not immediately apparent, taxation is a powerful concept. It is at the heart of some good heuristics for playing in a sum of games described in Chapter 5.

This section defines the *temperature* of a game, and then specifies a simple equation for the value of a game when a tax of t is imposed on every move.

2.6.1 Temperature

The temperature of a game, $\sigma(G)$, is the maximum tax that can be imposed and still have Left and Right willing to play first in G. For example, consider the game H_9 (Figure 2.7) and consider how the players react when a tax of t is imposed on every move for different values of t:

$t < 100$: Both players want to play first in H_9. That is, a player prefers paying t to his opponent and gaining 100 than having his opponent pay him t but losing 100.

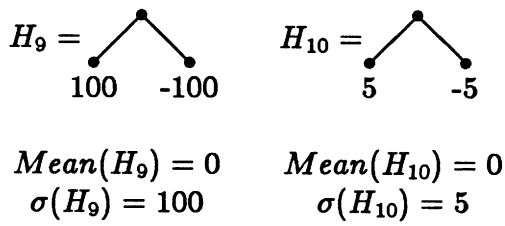


Figure 2.7: Defining H_9 and H_{10}

$t = 100$: Both players are indifferent; it does not matter who plays first.

$t > 100$: Neither player will want to play first.

Since the players are willing to pay a tax of $t \leq 100$ in order to play first in H_9, $\sigma(H_9) = 100$.

Intuitively, temperature measures how excited a player is to play in a game G. By quantifying how much a player is willing to pay in order to move in G, it provides some measure for the worth of a move in G. For example, consider the two games in Figure 2.7. If the two games appear in a sum of games, then both players will be very anxious to play first in H_9 but will be relatively indifferent as to who plays first in H_{10}. Temperature captures this idea, since $\sigma(H_9) \gg \sigma(H_{10})$.

Temperature measures the value of a move in a game. Mean value measures the inherent worth of the game having explicitly factored out the advantage a player gains for moving first in a game. Together, they provide a good first order approximation to a game.

2.6.2 The Value of a Taxed Game

The value of a game G for a player can be defined recursively as what his opponent gains after he has made one move, *i.e.*, if G is not a number then:

$$V_L(G) = V_R(L_1(G)) \tag{2.1}$$

$$V_R(G) = V_L(R_1(G)) \tag{2.2}$$

This subsection formulates similar equations for the value of a game when a tax of t is imposed. The following notation is used:

- ${}^tL_n(G) \Rightarrow$ The position that results after n moves when a tax of t is imposed on each move, players alternate moves, and both players play optimally.
- ${}^tR_n(G) \Rightarrow$ has the same meaning as ${}^tL_n(G)$ except that Right starts the play in G.
- ${}^tV_L(G) \Rightarrow$ is the final score when a tax of t is imposed on each move, players alternate turns, Left starts, and both players play optimally.
- ${}^tV_R(G) \Rightarrow$ is the same as ${}^tV_L(G)$ except that Right starts the play in G.

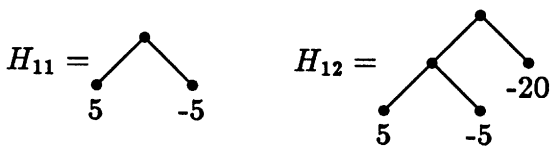
If $t > \sigma(G)$ then neither player is willing to move in G. In that case it is convenient to define ${}^tV_L(G)$ and ${}^tV_R(G)$ to be the same value as when $t = \sigma(G)$. If $t \leq \sigma(G)$, then the value for Left is the value for Right in ${}^tL_1G$ minus the tax t that Left paid to Right. Thus:

$$ {}^tV_L(G) = \begin{cases} {}^tV_R({}^tL_1G) - t & \text{if } t \leq \sigma(G) \\ {}^{\sigma(G)}V_L(G) & \text{if } t > \sigma(G) \end{cases} \tag{2.3} $$

where $\sigma(G)$ is the minimum t such that ${}^tV_L(G) = {}^tV_R(G)$. The equation for ${}^tV_R$ is the dual [1] of the above equation. For example, Figure 2.8 shows the values of ${}^tV_L(H_{11})$ and ${}^tV_R(H_{11})$ for different values of t.

Once the tax is set, it remains constant through out play in a game. However, not all moves in a game are worth the same amount. Hence, it is common for the players to be willing to pay the tax only for some portion of a game. For example, consider the game H_{12} shown in Figure 2.8. Assume that Left pays t and moves to $\{ 5 \mid -5 \}$. If $t \leq 5$, then Right will respond to Left's move. The taxes balance out, and the final score is -5. However, if $5 < t \leq 10$ then Right will not respond to Left's move. Since ${}^tV_R(\{ 5 \mid -5 \}) = 0$, the final score is $0 - t$.

[1] Since Left is trying to maximize and Right is trying to minimize, an equation for Left and can be converted into an equation for Right by taking its dual. The dual of an equation is obtained by transforming all $<$ to $>$, $\leq$ to $\geq$, $+$ to $-$, and $-$ to $+$. Furthermore, all Rs become Ls and all Ls become Rs in the notation ${}^tV_L, {}^tV_R, {}^tL_n$ and ${}^tR_n$.



$\sigma(H_{11}) = 5$ $\sigma(H_{12}) = 10$

	$t=0$	$t=2$	$t=4$	$t=6$	$t=8$	$t=10$
${}^tV_L(H_{11})$	5	3	1	0	0	0
${}^tV_R(H_{11})$	-5	-3	-1	0	0	0
${}^tV_L(H_{12})$	-5	-5	-5	-6	-8	-10
${}^tV_R(H_{12})$	-20	-18	-16	-14	-12	-10

Figure 2.8: Computing ${}^tV_L$ and ${}^tV_R$ for H_{11} and H_{12}

2.7 Thermographs

Thermographs provide a simple, efficient method for computing $\sigma(G)$, and for computing ${}^tV_L(G)$ and ${}^tV_R(G)$ for all values of t. A *thermograph* is a plot of ${}^tV_L$ and ${}^tV_R$ for increasing values of t where the tax is on the y axis, and the value of the game is on the x axis. In order to keep ${}^tV_L$ to the left of ${}^tV_R$, the values on the x axis are in decreasing order. For example, the thermograph for $\{\, 5 \mid -5 \,\}$ is shown in Figure 2.9.

Since taxing a number has no effect, the thermograph for a number is a vertical line. The thermograph for $G = \{\, L \mid R \,\}$ is obtained recursively from the thermograph of L and R by applying Equation 2.3 and its dual. The thermograph for G has three parts: its left edge, its right edge, and its mast. What follows is a description of how each part is obtained.

By Equation 2.3, when $t \leq \sigma(G)$:

$$ {}^tV_L(G) = {}^tV_R(L) - t. $$

Hence, the left edge of G's thermograph is obtained by subtracting t from every point on the right edge of L's thermograph. Graphically, subtracting increasing values of t corresponds to changing lines of slope one ($\diagdown$) into

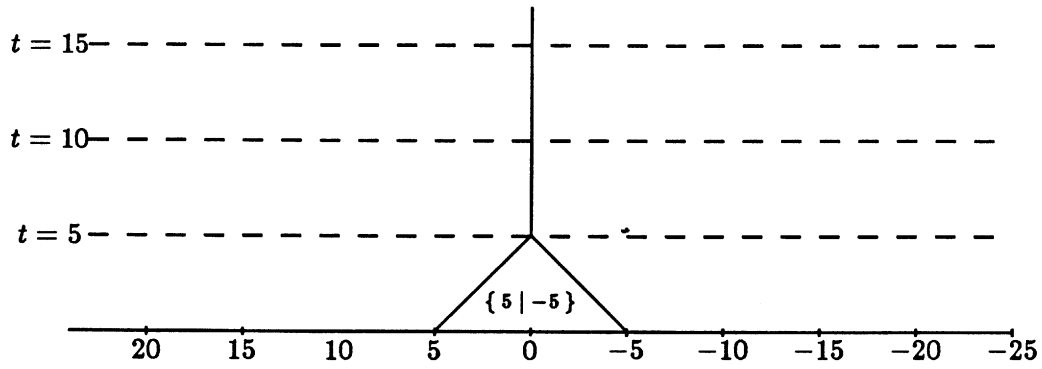


Figure 2.9: Thermograph for $\{ 5 \mid -5 \}$

vertical lines, and changing vertical lines into lines with slope minus one ($\diagup$).

Similarly, by the dual of Equation 2.3 when $t \leq \sigma(G)$, then

$$^tV_R(G) = {}^tV_L(R) + t.$$

Hence, the right edge of the thermograph for G is obtained by adding t to every point on the left edge of R's thermograph. Graphically, adding increasing values of t corresponds to changing lines of slope minus one ($\diagup$) into vertical lines, and changing vertical lines into lines with slope one ($\diagdown$).

The plot of ${}^tV_L(G)$ and ${}^tV_R(G)$ intersect at $\sigma(G)$. Since for all $t > \sigma(G)$:

$$^tV_L(G) = {}^tV_R(G) = {}^{\sigma(G)}V_L(G)$$

each thermograph is surmounted by an infinite vertical mast that starts at $t = \sigma(G)$.

To construct the thermograph for a game, start at the bottom of the tree and work up. For example, consider the game H_{13} shown in Figure 2.10. The thermograph for $R_1(H_{13})$ is obtained from the thermographs for $\{ 5 \mid -5 \}$ and -20 as shown in Figure 2.11. The thermograph for H_{13} is obtained from the thermographs for $R_1(H_{13})$ and $L_1(H_{13})$ as shown in Figure 2.12.

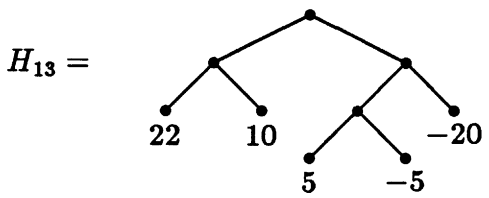


Figure 2.10: Defining H_{13}

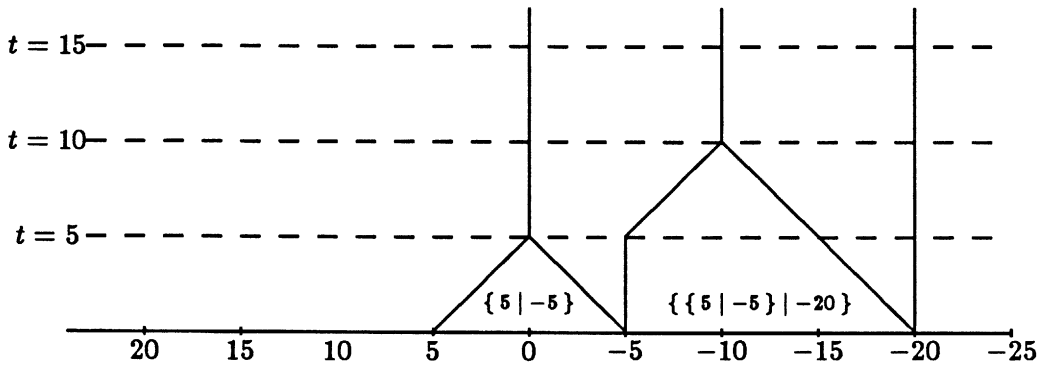


Figure 2.11: Constructing the Thermograph for $\{\{\ 5 \mid -5\ \} \mid 20\ \}$

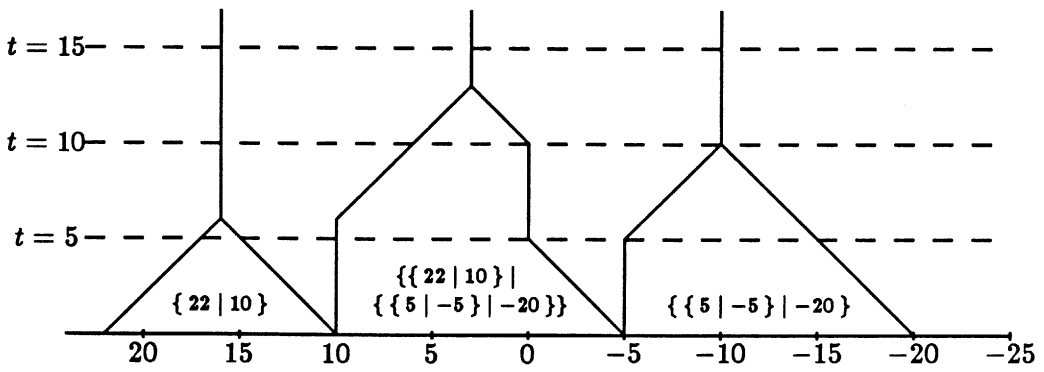


Figure 2.12: Constructing the Thermograph for H_{13}

The line representing ${}^tV_L(G)$ in a thermograph either goes vertically up or goes up and to the right (slope -1). The line representing ${}^tV_R(G)$ either goes vertical up or goes up and to the left (slope 1). Furthermore, since the lines meet when $t = \sigma(G)$, it is easy to show the following bounds on ${}^{\sigma(G)}V_R(G)$

$$ {}^tV_L(G) \geq {}^{\sigma(G)}V_L(G) \geq {}^tV_R(G) \tag{2.4} $$

$$ {}^tV_L(G) \leq {}^{\sigma(G)}V_L(G) \leq {}^tV_L(G) + t \tag{2.5} $$

2.8 Mean Value Revisited

One surprising fact is that the value of a game when a tax of $t \geq \sigma(G)$ is imposed equals the mean value of a game, *i.e.*,

$$ \forall t \geq \sigma(G) : Mean(G) = {}^tV_L(G) = {}^tV_R(G). \tag{2.6} $$

This result is proved by Hanner [7].

[2] contains a short argument for why Equation 2.6 is true. Unfortunately, the argument is unconvincing. The heart of the argument is that taxation is a linear function. However, the argument assumes that if in the game $A + B$ a tax $t : \sigma(A) < t < \sigma(B)$ is imposed on every move then play in the game is equivalent to play in the game:

$$ {}^{\sigma(A)}V_L(A) + B. $$

However, this assumes the result. If taxation is not linear, then one has now way of knowing what play in $A + B$ is like when a tax is imposed.

Conway[6] provides a more convincing argument. However, special care is still need to handle the case when t is close to $\sigma(A)$.

Since the value of a game when a tax of $t \geq \sigma(G)$ is imposed equals the mean value of a game, thermographs provide an efficient way of computing mean values. In particular, to compute the mean value of a game G, simply draw the thermograph for G and read off the graph ${}^tV_L$ for $t \geq \sigma(G)$. For example, the thermograph for H_{13} (Figure 2.12) shows that $mean(H_{13}) = {}^{\sigma(H_{13})}V_L(H_{13}) = 2.5$, and the thermograph for ${}^tR_1(H_{13})$ (Figure 2.11) shows that $mean({}^tR_1(H_{13})) = 10$.

Chapter 3

Easy Games - Hard Games

In general, SUM is $PSPACE$-complete. However, there exists restricted classes of SUM that have polynomial time solutions. This chapter analyzes a few restricted classes and shows that some have polynomial time solutions while others seem hard to play.

3.1 Switches

A game $\{ x \mid y \}$ where x, y are numbers and $x > y$ is called a *switch*. A sum of switches, though not hard to play, is an important special case that is used repeatedly through out this thesis.

Any switch $\{ x \mid y \}$ can be unbiased by converting it into

$$u + \{ v \mid -v \}$$

where u equals $1/2(x + y)$ and v, the temperature of the switch, equals $1/2(x - y)$. For example,

$$\{ 20 \mid -12 \} \Longrightarrow 4 + \{ 16 \mid -16 \}.$$

Left does not care if the game consists of $\{20 \mid -12\}$ or if the game consists of $\{ 16 \mid -16 \}$ but he is given 4 additional points at the start of the game.

In general, it is convenient to unbias a switch. Though the value of u affects the value of the final score, it does not affect optimal play and can be ignored in all discussions of strategy.

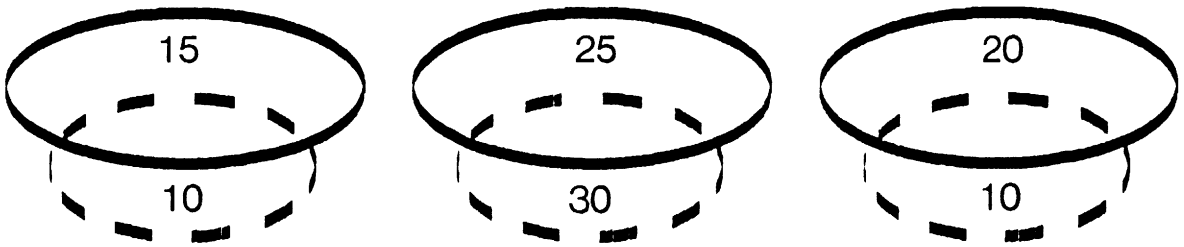


Figure 3.1: Stacked Coins

Consider the following game composed of n unbiased switches:

$$\{ x_1 \mid -x_1 \} + \cdots + \{ x_n \mid -x_n \}$$

where $x_1 \geq \ldots \geq x_n$. [2] makes a good analogy between this game and a game where a bunch of coins $(x_1, \ldots, x_n)$ are placed on a table. On each move a player can take a coin off the table and place it in his pocket. The final score represents the difference between the amount of money in Left's pocket and the amount in Right's pocket.

The optimal strategy for both players is to choose the largest switch (coin). If Left plays first, then the value of the game is equal to the alternating sum:

$$ALT = x_1 - x_2 + \cdots \pm x_n.$$

If Right plays first, then the value of the game is the negative of ALT.

3.2 Stacks of Coins

One simple generalization of the above game is to imagine several stacks of coins placed on a table as in Figure 3.1. On each turn a player can remove one of the exposed coins and place that coin in his pocket. The score is the difference between the amount of money in Left's pocket and the amount in Right's pocket.

The tree representation of a stack of two coins, x and y where x is on y, is shown in Figure 3.2. If $x < y$, then the game is a number. For example, in the game shown in Figure 3.1, Left is unwilling to take the 25 coin since

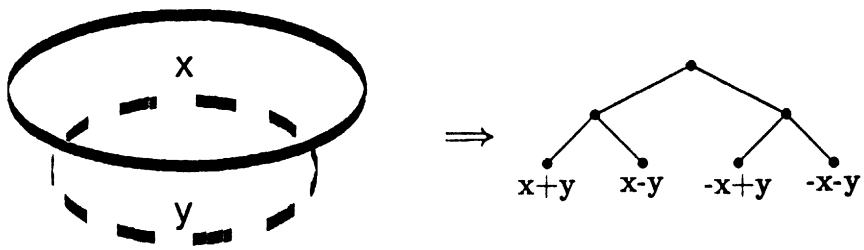


Figure 3.2: Representing a Stack of Coins

Right would then be able to take the 30 coin. From Left's point of view this is a net loss of 5.

Like the previous game, this game is easy to play. The optimal strategy is simply the greedy strategy, *i.e.*, take the largest coin on a stack which is not a number. For example, in the game shown in Figure 3.1 Left's optimal move is to take the 20 coin on top of the third stack.

3.3 Left Heavy Games

A *left heavy game* is a game of the form $\{\{x \mid y\} \mid z\}$ where x, y, z are numbers and $x \geq y \geq z$. A left heavy game is the simplest game for which the notion of a *sente* move for Left applies, *e.g.*, Figure 3.3. Hence, they add a good deal of complexity to a sum of games.

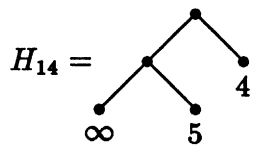
A *$f(n)$-left heavy game* is a sum composed n games where $f(n)$ of them are left heavy games and the rest are switches. The goal of this section is to analyze such games for various functions $f(n)$.

3.3.1 C-Left Heavy Games

A c-left heavy game, for any constant c, is handled efficiently by computing a formula for the score of the game. Without loss of generality, assume that the switches in a c-left heavy game have been unbiased. Then a c-left heavy game has the form:

$$\{\{x_1 \mid y_1\} \mid z_1\} + \cdots + \{\{x_c \mid y_c\} \mid z_c\} + \{t_1 \mid -t_1\} + \cdots + \{t_{n-c} \mid -t_{n-c}\}$$

where $t_1 \geq \ldots \geq t_{n-c}$.



After Left moves in H_{14}, he threatens to take ∞. Right is compelled to respond to prevent the threat. Hence, Left's move is *sente.*

Figure 3.3: A Left Heavy Tree where Left's Move is *Sente*

Theorem 3.1 *In a c-left heavy game, it is at least as good if not better for Left to play in the largest switch then to play in any other switch.*

Proof: Let G be a c-left heavy game, and let G^i be the game that results when Left moves in the i^{th} switch. The theorem states that from Left's point of view G^1 is as least as good as G^i.

It is sufficient to show that in $G^1 + (-G^i)$ Left has a winning strategy even when Right starts [5, page 78]. Consider $G^1 + (-G^i)$ with the j^{th} component game of G^1 paired with j^{th} component game of $-G^i$, *i.e.,*

$$
\begin{array}{lcll}
\{\{x_1 \mid y_1\} \mid z_1\} & + & -\{\{x_1 \mid y_1\} \mid z_1\} & + \\
 & & \vdots & \\
\{\{x_c \mid y_c\} \mid z_c\} & + & -\{\{x_c \mid y_c\} \mid z_c\} & + \\
t_1 & + & -\{t_1 \mid -t_1\}) & + \\
\{t_2 \mid -t_2\} & + & -\{t_2 \mid -t_2\} & + \\
 & & \vdots & \\
\{t_{i-1} \mid -t_{i-1}\} & + & -\{t_{i-1} \mid -t_{i-1}\} & + \\
\{t_i \mid -t_i\} & + & -t_i & + \\
\{t_{i+1} \mid -t_{i+1}\} & + & -\{t_{i+1} \mid -t_{i+1}\} & + \\
 & & \vdots & \\
\{t_n \mid -t_n\} & + & -\{t_{n-c} \mid -t_{n-c}\} &
\end{array}
$$

Except for the $(c+1)^{th}$ and $(c+1+i)^{th}$ component games, each component game is paired with its negative. Whenever Right plays in such a pair,

Left can respond with the negative of Right's move in the other game, and cancel any gain Right makes by going first in the pair.

Hence, the problem is reduced to showing that Left has a winning strategy when Right moves first in the following game:

$$t_1 + (-\{ t_1 \mid -t_1 \}) + \{ t_1 \mid -t_1 \} - t_i$$

Since the negative of an unbiased switch is the switch itself, the game above is equivalent to:

$$t_1 + \{ t_1 \mid -t_1 \} + \{ t_1 \mid -t_1 \} - t_i$$

Right's optimal strategy is to play in the largest switch, $\{ t_1 \mid -t_1 \}$, and Left's optimal response is to play in the remaining switch, $\{ t_i \mid -t_i \}$. The final score is $t_1 - t_1 + t_i - t_i = 0$ with Right to move. Thus Left wins. ∎

Using the above theorem, the optimal strategy for a c-left heavy game, for any value of c, can be computed. However, for simplicity, only the case when $c = 1$ is considered here.

Given Theorem 3.1, optimal play in a 1-left heavy game proceeds in three distinct stages. First, players alternately play in the largest switch. At some point, a player will play in the left heavy tree. If Left plays there, the tree is reduced to a switch. If Right plays there, the tree is reduced to a number. In either case, the players then return to playing alternately in the largest switch.

The key question is on what move should Left or Right play in the left heavy tree. To answer this question it is sufficient to compute formulas for the score of the game when Left and Right move in the left heavy tree at move i.

It is convenient to have a shorthand notation for the amount the switches $\{ t_i \mid -t_i \}$ through $\{ t_j \mid -t_j \}$ add to the score when Left starts play in the game

$$\{ t_1 \mid -t_1 \} + \ldots + \{ t_{n-c} \mid -t_{n-c} \}$$

In particular, let:

$$[i, j] = (t_i - t_{i+1} + \ldots \pm t_j) * (-1)^{i+1}$$

If Right moves first in the left heavy tree at move i, then the score for the first stage of the game is [1,i-1]. For the second stage, right moves in

the left heavy tree and takes z. For the third stage, play resumes with Left and hence the score is the negative of the alternating sum of switches from i to n. Adding together the three stages and simplifying, we get:

$$\text{RightScore(i)} = [1, n] + z - 2[i, n]$$

When Left plays in the left heavy game, he throws a biased switch, $\{ x \mid y \}$, into the set of remaining unbiased switches. It is convenient to view that switch as unbiased, *i.e.*, $\{ x \mid y \} = m_0 + \{ t_0 \mid -t_0 \}$ where $m_0 = (x+y)/2$ and $t_0 = (x-y)/2$. It is also convenient to know where the unbiased switch $\{ t_0 \mid -t_0 \}$ fits into the ordered list of biased switches. Let k be the number such that $t_k \geq t_0 \geq t_{k+1}$.

If Left first moves in the left heavy tree at move $i > k$, then Right will immediately respond in the $\{ x \mid y \}$ switch and play will then proceed as usual. So,

$$\text{LeftScore(i)} = [1, n] + m_0 - t_0 \text{ when } i > k.$$

If Left first moves in the left heavy tree at move $i \leq k$, then the score for stage 1 is [1,i-1]. No score is generated for the second stage of the game, but the unbiased switch $\{ t_0 \mid -t_0 \}$ is thrown into the game. The score for the third stage is: $-[i, k] + m_0 + (-1)^{k+1} t_0 + [k+1, n]$. Adding together the score at each stage and simplifying produces:

$$\text{LeftScore(i)} = [1, n] - 2[i, k] + m_0 + (-1)^{k+1} t_0 \text{ when } i \leq k.$$

Assume that Left always goes first. For increasing i, (2 4 6 ...) the value of RightScore(i) will decrease. Since Right is trying to minimize the score, Right will delay moving in the left heavy tree. However, if on the next move it is to Left's advantage to play in the left heavy tree, then Right will play there in defense. Specifically, Right moves in the left heavy tree on move i iff

$$\text{RightScore(i+2)} \leq \text{LeftScore(i+1)}$$

Similarly, Left's optimal strategy on move i is to move in the left heavy tree iff:

$$\text{LeftScore(i+2)} \geq \text{RightScore(i+1)}$$

For example, consider the game shown in Figure 3.4. It is easy to

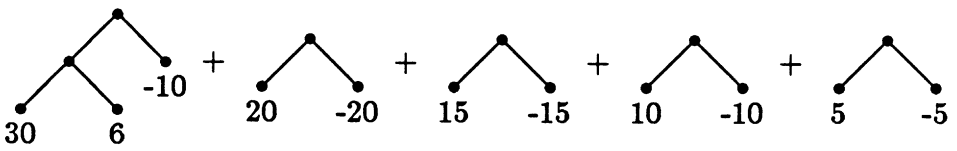


Figure 3.4: A 1-Left Heavy Game

compute that:

$$\begin{aligned} LeftScore(1) &= 6 \\ RightScore(2) &= 20 \\ LeftScore(3) &= 16 \\ RigthScore(4) &= 10 \\ LeftScore(5) &= 16 \end{aligned}$$

Left can play in the largest switch on his first move but must move in left heavy tree on the third move.

3.3.2 Log(n)-Left Heavy Games

A log(n)-left heavy game can be played in polynomial time, though not as efficiently as a c-left heavy game.

Under reasonable play, the log(n)-left heavy game can take on only a polynomial number of different states. In particular, each of the log(n) left heavy games can be in one of two states, *i.e.*, either $\{ \{ x \mid y \} \mid z \}$ or $\{ x \mid y \}$. Furthermore, by Theorem 3.1, reasonable play in the switches will consist of first playing out the switches in order. So there are only n different ways in which the switches will appear. Hence, there are

$$2^{\log(n)} * n$$

possible configurations.

Since there are only polynomial number of different states, standard dynamic programming techniques can be used to solve the problem in polynomial time.

3.3.3 n-Left Heavy Games

There is no known polynomial time algorithm for determining the optimal strategy for n-left heavy games. This section will present some properties of left heavy games that suggest that finding a polynomial time strategy is hard.

In order to study the properties of left heavy games it is convenient to consider a special class of left heavy games that have the form $\{\{x \mid 0\} \mid z\}$. This can be done without loss of generality since any left heavy game can be put into this form via the following transformation:

$$\{\{x \mid y\} \mid z\} = -y + \{\{x - y \mid 0\} \mid z - y\}$$

The value $-y$ will not affect the strategy of either player and can be ignored.

The following notation is used:

$$A \trianglerighteq_L B \text{ iff an optimal move for Left in } A + B \text{ is in } A$$

It is easy to compute that if $A = \{\{a \mid 0\} \mid c\}$ and $B = \{\{d \mid 0\} \mid f\}$ then

$$A \trianglerighteq_L B \iff min(f + a, 0) \geq min(c + d, 0)$$

In the best of all possible worlds, there would exist an evaluation function F such that $A \trianglerighteq_L B$ iff $F(A) \geq F(B)$. However, for this to be true $\trianglerighteq_L$ would have to be transitive between left heavy games. That is not the case. For example,

$$\{\{1 \mid 0\} \mid -1\} \trianglerighteq_L \{\{2 \mid 0\} \mid -1\} \trianglerighteq_L \{\{2 \mid 0\} \mid -2\}$$

but it is not the case that

$$\{\{1 \mid 0\} \mid -1\} \trianglerighteq_L \{\{2 \mid 0\} \mid -2\}$$

Even given that $\trianglerighteq_L$ is not transitive, it is natural to conjecture that if $A \trianglerighteq_L B$, and $A \trianglerighteq_L C$ then in the sum $A + B + C$ the optimal move is A. However, this is not true. For example, consider the games shown in Figure 3.5. In the sum of any pair containing H_{15}, Left's optimal move is in H_{15}. However, in the sum $H_{15} + H_{16} + H_{17}$, Left's optimal move is in H_{16}.

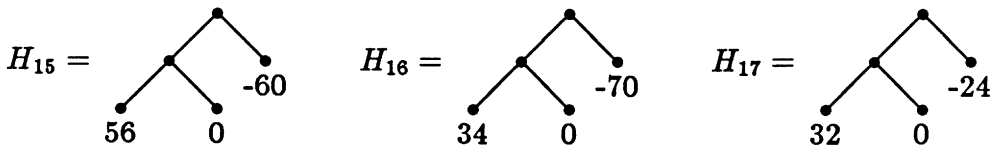


Figure 3.5: Defining H_{15}, H_{16}, and H_{17}

From Right's point of view, the world is a bit simpler. It is easy to show that if $A = \{\, \{\, a \mid 0 \,\} \mid c \,\}$ and $B = \{\, \{\, d \mid 0 \,\} \mid f \,\}$ then

$$A \trianglerighteq_R B \iff c \geq f$$

The $\trianglerighteq_R$ operator is transitive for Right. However, $A \trianglerighteq_R B$ and $A \trianglerighteq_R C$ does not imply that A is the optimal move for Right in the game A+B+C. For example:

$$\{\, \{\, 4 \mid 0 \,\} \mid -61 \,\} \trianglerighteq_R \{\, \{\, 73 \mid 0 \,\} \mid -66 \,\},$$

and

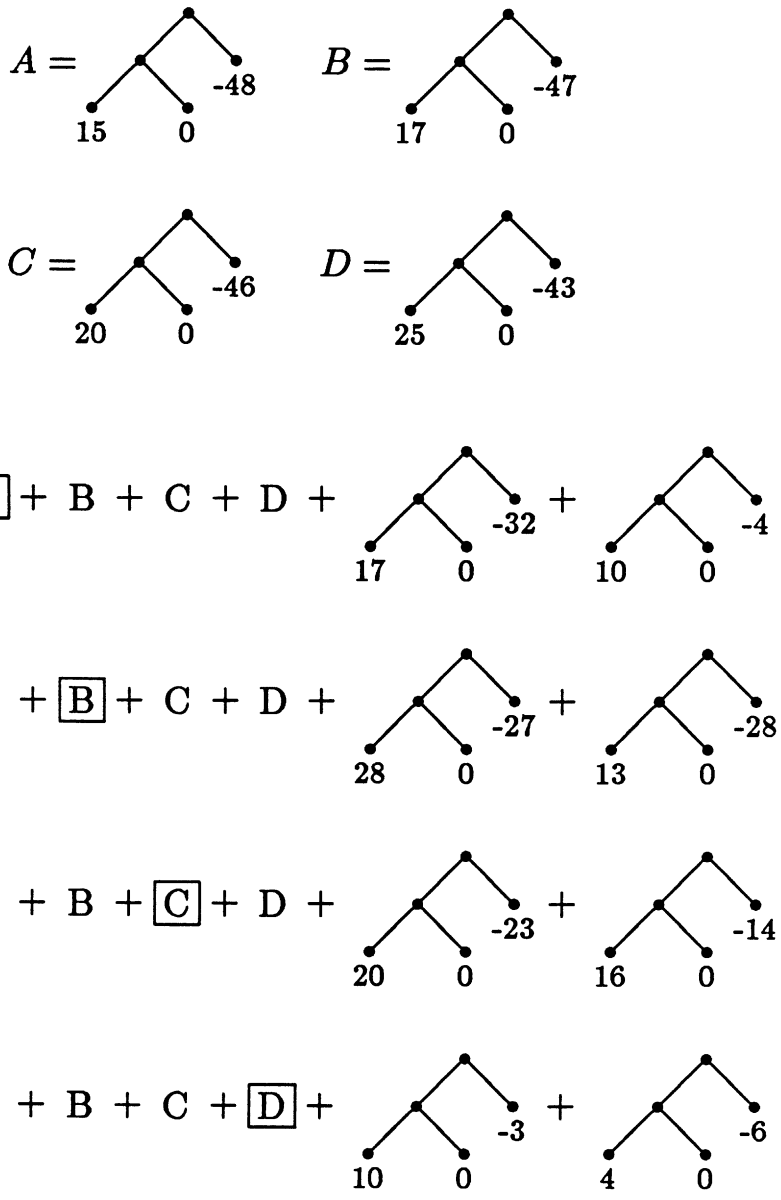
$$\{\, \{\, 4 \mid 0 \,\} \mid -61 \,\} \trianglerighteq_R \{\, \{\, 18 \mid 0 \,\} \mid -95 \,\},$$

but in the sum:

$$\{\, \{\, 4 \mid 0 \,\} \mid -61 \,\} + \{\, \{\, 73 \mid 0 \,\} \mid -66 \,\} + \{\, \{\, 18 \mid 0 \,\} \mid -95 \,\}.$$

Right's optimal move is in the second game.

All of the above examples show that understanding the context in which two or more a left heavy games appear is paramount to understanding the relative importance of the games. On further example is shown in Figure 3.6. The left heavy games A,B,C, and D, are placed in four different contexts. In the first sum, the optimal move is in game A. In the second sum, the optimal move is in game B. In the third sum, the optimal move is in game C. In the fourth sum, the optimal move is in game D.



The optimal move in each game appears in the box

Figure 3.6: The Importance of Context

Chapter 4

Complexity of SUM

Lockwood Morris[17] shows that determining optimal play in a disjunctive sum of games is $PSPACE$-complete. His proof holds when the component games in the sum are of depth 4 or more, but it leaves open the possibility that instances of SUM composed of shorter games could (even assuming $P \neq PSPACE$) be handled in polynomial time.

This chapter narrows the gap between those problems known to be in P, and those problems known to be $PSPACE$-complete. Let $SUM_{d \leq n}$ refer to the problem of determining optimal play in a sum of games, where each component game has depth less than or equal to n. The natural question arises as to how small n can be and still have $SUM_{d \leq n}$ be $PSPACE$-complete. This chapter will prove that $SUM_{d \leq 2}$ is $PSPACE$-complete.

The result is important for two reasons. Previously, the only known way of determining the optimal play in an instance of $SUM_{d \leq 2}$ or $SUM_{d \leq 3}$ was via an exponential time search. However, there was no convincing evidence that the problem was hard enough to warrant such an algorithm Now, with a free conscience, one can accept an algorithm with an exponential running time.

The result also affects the types of heuristic solutions that are possible. The next chapter presents a relatively complex heuristic that determines a good move in a sum of games by using taxation to approximate each component game. One could imagine a simpler algorithm that approximated each component game with a depth two or depth three game. Since a depth two or a depth three game maintains the notion of *sente*, the approximation could be quite accurate. However, since $SUM_{d \leq 2}$ is $PSPACE$-complete

this approach will fail. The approximated version of the game will be (roughly speaking) as hard to solve as the original version.

What follows is the proof that $SUM_{d\leq 2}$ is $PSPACE$-complete. The proof proceeds in two parts. The first part proves that $SUM_{d\leq 2}$ is as hard as SAT, *i.e.*, the problem of determining if a boolean formula is satisfiable. The second part, expands upon the first part to show that $SUM_{d\leq 2}$ is equivalent to QBF, *i.e.*, the problem of determining if a quantified boolean formula is satisfiable.

4.1 $SUM_{d\leq 2}$ is NP-hard

Morris transforms an arbitrary instance of PARTITION consisting of numbers $x_1, \ldots, x_n$ and the value $S = 1/2 \sum x_i$ into an instance of SUM. He shows that under optimal play, Left chooses a subset of the $x's$, whose sum is L, to remain in play. Right then has the option of playing such that the final score is either $S - L$ or $L - S$. Left wins iff either of Rights options result in a final score that is greater than or equal to zero. Hence, Left wins iff $S = L$ and he has partitioned the x's.

This section will prove that $SUM_{d\leq 2}$ is NP-hard. The proof given here is very similar to Morris's proof. However, the following proof reduces the size of the component games by dealing with alternating sums of numbers. This allows the more complex games used by Morris to be replaced by switches.

The proof proceeds in three steps. First it is shown that ALT is NP-complete. Namely, given the set of integers and a value, the problem of deciding if there is a subset of integers such that their alternating sum (taken in descending order) equals the given value is NP-complete.

Next, SUBSET SWITCH is shown to be NP-complete. Namely, given a value, B, and a set of switches, the problem of deciding if there exists a subset of switches, X, such that $V_L(X) = B$ [1] is NP-complete.

Finally, $SUM_{d\leq 2}$ is shown to be NP-hard. An arbitrary instance of SUBSET SWITCH, consisting of a set of switches and a value B, is trans-

[1] V_L is defined in Section 2.1. When applied to a set, it is the final score of the sum composed of all elements in the set assuming that Left starts and both players play optimally.

formed into $SUM_{d\leq 2}$. A game is constructed where optimal play proceeds as follows: Left plays and leaves a subset of switches, X, such that $V_L(X) = L$. Right then has the choice of playing such that the final score is $B - L$ and $L - B$. Left wins if both of Right's alternatives results in a final score that is greater than or equal to zero. So, Left wins iff he has been able to solve the SUBSET SWITCH problem.

Lemma 4.1 *ALT is NP-complete. Given the set Y of integers and a value B, the problem of deciding if there exists a subset $Y' = \{y'_0, y'_1, \ldots, y'_k\} \subset Y$ where $y'_0 \geq y'_1 \geq \ldots \geq y'_k$ and*

$$B = y'_0 - y'_1 + \cdots \pm y'_k$$

is NP-complete.

Proof: ALT is in NP. A non-deterministic algorithm simply guesses a subset Y' and checks in polynomial time that its alternating sum is equal to B.

Consider the PARTITION problem. An instance consists of a set of integers $X = \{x_0, \ldots, x_n\}$ where $x_0 \geq \ldots \geq x_n$. Karp[10] showed that problem of determining if there is a subset $X' \subseteq X$ such that

$$\sum_{x_i \in X'} x_i = \sum_{x_i \in X - X'} x_i$$

is NP-Complete.

It is sufficient to reduce PARTITION to ALT. In particular, the following construction creates an an instance of ALT consisting of a value B and a set of integers Y such that it can be solved iff the given instance of PARTITION can be solved:

1. $B \leftarrow \frac{1}{2} \sum x_i$
2. For each integer $x_i \in X$, add to the set Y the integers y_{i1} and y_{i2}. Assume k is the maximum number of bits required to represent B and x_0. Let y_{i1} and y_{i2} be s bits long where $s = 2n + \log(n) + k$ bits. Then:

$$\begin{aligned} y_{i1} &= 2^{s-2i} + x_i \\ y_{i2} &= 2^{s-2i}. \end{aligned}$$

	$2n$						$\log(n)$			k
y_{01}:	1	0	0	0	$\cdots$	0	0	$\cdots$	0	x_1
y_{02}:	1	0	0	0	$\cdots$	0	0	$\cdots$	0	0
y_{11}:	0	0	1	0	$\cdots$	0	0	$\cdots$	0	x_2
y_{12}:	0	0	1	0	$\cdots$	0	0	$\cdots$	0	0
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
y_{n1}:	0	0	0	0	$\cdots$	1	0	$\cdots$	0	x_n
y_{n2}:	0	0	0	0	$\cdots$	1	0	$\cdots$	0	0

Figure 4.1: Constructing an Instance of ALT

The lower order k bits of y_{i1} contain the value x_i, and the higher order bits are 0 except for the $(s-2i)^{th}$ bit. All bits in Y_{i2} are zero except for the $(s-2i)^{th}$. Figure 4.1 shows the construction of the y_{i1}, y_{i2} pairs. Note, $\log(n)$ bits are left between the high order positional bits and the lower order bits containing x_i to guard against overflow.

It is important to note that the y's decrease in size rapidly, and that they are all significantly larger than the value of B. If y_{i1} is added to the alternating sum, then to reach the value of B it is necessary to subtract the value y_{i2}. Any solution of the instance of ALT constructed above either contains both y_{i1} and y_{i2} or contains neither value. Furthermore, since $x_i = y_{i1} - y_{i2}$ the value of the alternating sum of the y_{i1}, y_{i2} pairs is the sum of the x_i's coded in the lower k bits of the y_{i1} values.

The given instances of ALT and PARTITION are equivalent problems. The solution for one implies the solution for the other. A solution for the instance of ALT constructed above will contain a number of y_{i1}, y_{i2} pairs. The corresponding solution to PARTITION is the set of x_i values coded in the k lower order bits of the y_{i1}'s. Similarly, given a solution, X', for an instance of PARTITION, the corresponding solution to ALT is the set of y_{i1}, y_{i2} values constructed from the $x_i \in X'$.

Since the construction is accomplished in polynomial time, and since ALT can be solved if and only if PARTITION can be solved, ALT is NP-complete. ∎

Corollary 4.1 *SUBSET SWITCH is NP-complete. In particular, given a value S and a set of unbiased switches the problem of deciding if there exists a subset of switches whose sum equals B is NP-complete.*

Proof: The value for Left in a sum of unbiased switches is the alternating sum of the component values of the switches. For example:

$$V_L(\{\,9 \mid -9\,\} + \{\,4 \mid -4\,\} + \{\,3 \mid -3\,\} + \{\,2 \mid -2\,\}) = 9 - 4 + 3 - 2$$

The problem of choosing a subset of switches such that the value for Left when Left plays first in their sum, *i.e.*, the alternating sum of their components, equals a given value is equivalent to the ALT problem.

Theorem 4.1 *$SUM_{d\leq 2}$, i.e., determining the optimal play in a sum of games where each component game has depth less than or equal to 2, is NP-hard.*

Proof: Transform an arbitrary instance of SUBSET SWITCH, consisting of a value B and a set of switches $\{\,x_1 \mid -x_1\,\},\ldots,\{\,x_n \mid -x_n\,\}$ where $x_1 \geq \ldots \geq x_n$, into an instance of $SUM_{d\leq 2}$. Namely, construct the game:

$$J + G_{11} + G_{12} + \cdots + G_{n1} + G_{n2} + H + I + \uparrow$$

where

$$\begin{array}{lcl}
J & \rightarrow & \{\{\infty \mid \sum_{i=1}^{n} A_i - C\} \mid -\infty\} \\
G_{i1} & \rightarrow & \{\{x_i \mid -x_i\} \mid -A_i\} \\
G_{i2} & \rightarrow & \{0 \mid -A_i\} \\
H & \rightarrow & \{\,C \mid -C\,\} \\
I & \rightarrow & \{D \mid -S, \{S \mid -E\}\} \\
\uparrow & \rightarrow & \{0 \mid \{0 \mid 0\}\} \\
A_i & > & (4 * A_{i+1}) \\
A_n & > & 4 * (C + D + E + S + \sum_{i=1}^{n} x_i) \\
C & > & D + \sum_{i=1}^{n} x_i \\
D & > & 4 * E \\
E & > & 2 * \sum_{i=1}^{n} x_i
\end{array}$$

In the above sum, the game $\uparrow$ will never affect the score. Furthermore since neither player can gain an advantage by playing in it, it will not be

played until all other games have ended. Thus, it will not affect either player's strategy and will be ignored in all discussions of strategy.

However, $\uparrow$ does affect the above game. Left wins a sum of games if the final score is greater than zero, or if the final score equals zero and it is Right's turn to play when the game ends. $\uparrow$ guarantees that when the game ends, it is Right's turn to move. Thus, a final score of zero will be a victory for Left.

To show that Left has a winning strategy iff he can choose a subset of switches whose value is S, it will be convenient to first consider how the players are expected to play. This will be called the *normal* strategy. Afterwards it will be shown that neither player can hope to benefit from deviating from the *normal* strategy.

Normal play proceeds in three stages: beginning, middle and final. The beginning stage consists of Left playing in J and Right responding in the same game, picking up $\sum A_i - C$.

The middle stage consists of Left and Right both playing in the G games. Left begins by playing in either G_{11} or G_{12}. Right responds in the other game. Left then chooses to play in either G_{21} or G_{22} and Right responds in the other one. Play proceeds down the pairs of G_i's until all the G games are played. This ends the middle stage.

Notice that Left's choice between G_{i1} and G_{i2} is the choice between including or excluding the switch $\{ x_i \mid -x_i \}$ in the final game. In normal play Left has total control over which switches are included and which are excluded.

When the final stage begins, the game has been reduced to the sum of I, H, and the subset of switches that Left choose to leave in the game. For convenience, Let L be V_L of the sum of those switches.

Play begins in the final stage with Left playing in H.

Right then has a major decision to make. He must choose between playing in the first option and playing in second option of I. If he plays in the first option $(-S)$, Left will start the play in the remaining switches. The switches will be played out and the final score will be $-S + L$. If he plays in the second option $(\{ S \mid -E \})$, Left will respond in $\{ S \mid -E \}$ taking S and Right will start the play in the switches. The final score will be $S - L$.

Left will win iff both of Right's options (taking $S - L$ or $L - S$) result

in a non-negative final score. This is only true if Left solved the SUBSET SWITCH problem.

To show that the *normal* play described above is optimal for both players, it will be shown that in each stage of play neither player can gain an advantage by deviating from the strategy described above.

The play in the beginning stage of the game is obviously optimal. If Left does not move in J, Right will gain $-\infty$ and win. Similarly, after Left plays in J, he is threatening to gain ∞. Right must respond in the same game to prevent this threat.

To analyze the play in the middle stage of the game, note that after the beginning stage, Left has a huge advantage $(\sum A_i - C)$. With normal play Right recoups his loss by gaining one $-A_i$ for each of the (G_{i1}, G_{i2}) pair. For Right, the major drive in the middle stage is his need to regain at least $-\sum A_i$.

If Left is playing in the normal way, can Right gain an advantage by not responding directly to Left's moves in the pair G_{i1}, G_{i2}? The answer is no. If Right plays elsewhere, Left will play in the other G_i game preventing Right from getting $-A_i$. This is bad for Right since:

$$A_i > (2 * \sum_{k=1}^{i-1} A_k) + C + D + E + S + \sum_{i=1}^{n} x_i$$

Even if Right could play first in all the other games, he would gain less than A_i. It is not only better for Right to play in G_i than to play somewhere else, it is better for Right to play in G_i than to play everywhere else.

Note that the above argument is symmetrical. If for some reason, Right was the first player to play in pair G_{11}, G_{12}, then Left would be forced to respond by playing in the other G_1 game. If Left did not do so, he would be unable to compensate for Right having gained $2 * -A_1$. The fact that a move in the pair G_{11}, G_{12} forces the opponent to respond in that pair of games is the key fact needed to prove that Left can not gain an advantage by deviating from normal play.

In particular, suppose that Left and Right have played in the normal way in the games: $(G_{11}, G_{12}, \ldots, G_{i-1,1}, G_{i-1,2})$ and suppose that Left decided to play somewhere besides in the pair G_{i1}, G_{i2}. There are three cases that must be analyzed.

First suppose that Left decided to play in the G_{k1}, G_{k2} pair where $k > i$. It is easy to see that playing in the G_{k1}, G_{k2} pair early does not gain Left any advantage but only increases Right's options. Right can respond in the pair G_{i1}, G_{i2}. By the previous argument, Left is forced to respond in the same pair. Right can then play in the pair $G_{i+1,1}, G_{i+1,2}$ forcing Left to respond. Right can continue to force the play until the G_{k1}, G_{k2} pair is reached. Hence, Right can force a line of play that is equivalent to the normal play except that Right had control over which of the switches $\{ x_j \mid -x_j \}$ where $i \geq j > k$ were included in the remaining game. Left gains nothing by giving up his control.

The second case is that Left deviated from normal play in order to play in H early. Using the same reasoning as above, Right can force a line of play that is equivalent to the normal play except that Right has control for the pairs G_{i1}, G_{i2} through G_{n1}, G_{n2}. Left gains nothing by giving up control in this way.

The third possibility, is that Left deviates from normal play in order to play in I, $\uparrow$, or one of the $\{ x_i \mid -x_i \}$ switches. This is truly disastrous for Left. Not only does Right gain control for the pairs G_{i1}, G_{i2} through G_{n1}, G_{n2}, Right also gains the option of playing in H and gaining $-C$. Even if Left could then play first in every other game, he could not recoup his loss since $C > D + \sum x_i$.

When the final stage of the game commences, its Left's turn to play and the game has been reduced to the sum of H, I, $\uparrow$, and a subset of the switches.

With normal play Left would play in H and gain C. The only reason for Left not playing in H and allowing Right to play there, is that he hopes to gain more than C. However, C is greater than the sum of all of Left's options in all the remaining games, *i.e.*, $C > D + \sum x_i$. Not only is playing in H better than playing somewhere else, playing in H is better than playing everywhere else.

Normal play then dictates that Right plays in I. If Right plays in any other game, Left will play in I and pick up $D > 8 * \sum x_i$. Even if Right could play first in every other game, he would not gain more than D. Hence, playing in I is optimal.

After I has been played, the game is a sum of switches. The optimal strategy is simply the optimal strategy for playing in a sum of switches.

This is equivalent to the normal play described above.

Thus, normal play does describe the optimal play for both players. Left can win iff he can solve the SUBSET SWITCH problem. ▮

4.2 $SUM_{d\leq 2}$ is $PSPACE$-complete

This section will prove that $SUM_{d\leq 2}$ is $PSPACE$-complete. Stockmeyer and Meyer[27] proved that QBF, the problem of determining if a quantified boolean formula is true, is $PSPACE$-complete. This section will reduce QBF to $SUM_{d\leq 2}$.

Again, the proof is very similar to Morris's proof that SUM is $PSPACE$-complete[17], but alternating sums of numbers are used in the reductions. This allows the complex games used by Morris to be replaced by switches. Hence, the depth of the component games is reduced.

The proof consists of reducing QBF to 2P-EXACT-COVER, which is reduced to 2P-GEN-PARTITION, which is reduced to 2P-ALT, which is reduced to 2P-SUBSET-SWITCH which is reduced to $SUM_{d\leq 2}$. This is a rather long series of reductions. However, each step is analogous to a step in the NP-hardness proof given in section 4.1. The major difference between the two proofs is that the NP-harness proof reduces SAT to $SUM_{d\leq 2}$ via a series of one person games, where as this proof reduces QBF to $SUM_{d\leq 2}$ via a series of two person games. The structural similarity of the two proofs is shown in Figure 4.2.

The games used in the $PSPACE$-completeness proof (2P-EXACT-COVER, 2P-ALT, 2P-GEN-PARTITION, 2P-SUBSET-SWITCH) are the two person versions of the games used in the NP-hardness proof. They all have the same basic form. An instance of the game consists of the following set of objects:

$$X_1, \overline{X}_1, \cdots, X_n, \overline{X}_n, Y_1, \cdots, Y_m$$

and possibly the value B. Play begins with Left selecting either X_1, or $\overline{X}_1$ Right then selects either X_2, or $\overline{X}_2$. Left and Right continue to alternate turns until all the X objects have been selected. Left then selects some number of the Y objects. Left wins iff all the selected objects satisfy some condition.

The only difference between the games is what the objects are, and what

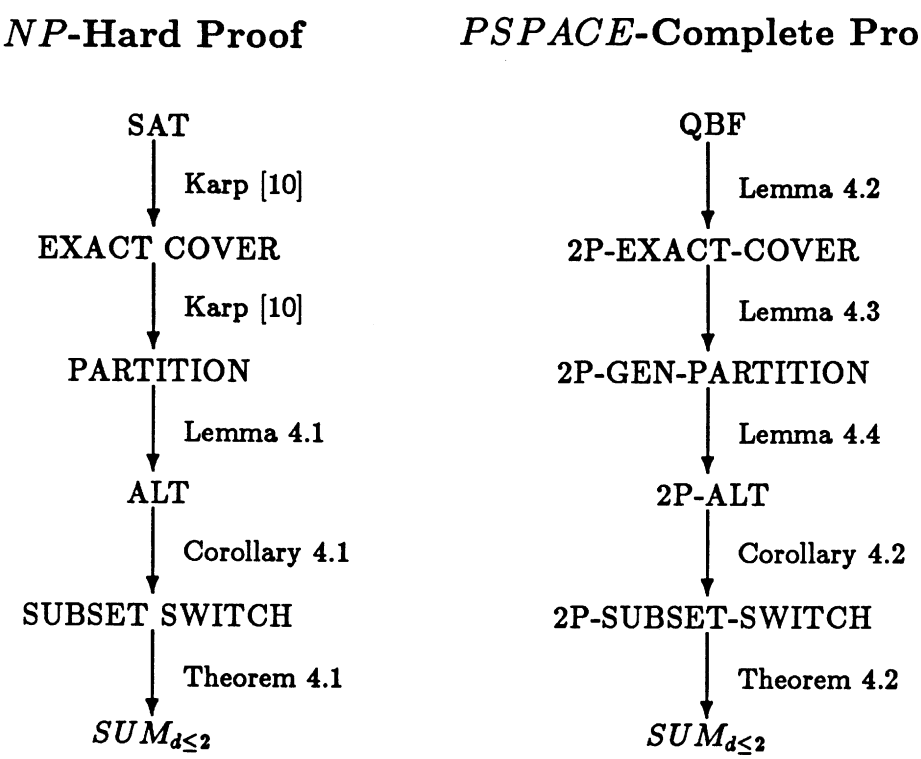


Figure 4.2: Comparing the Structure of the Proofs

the winning condition is. In 2P-EXACT-COVER the objects are subsets of a set, and Left wins iff the selected subsets form an exact cover of the set. In 2P-GEN-PARTITION the objects are numbers, and Left wins iff the sum of selected numbers equals B [2]. In 2P-ALT the objects are numbers, and Left wins iff the alternating sum of the selected numbers equals B. Finally, in 2P-SUBSET-SWITCH the objects are switches, and Left wins iff V_L of the sum of the selected switches equals B.

What follows are the reductions used to prove that $SUM_{d\leq 2}$ is $PSPACE$-complete. The first two reductions (QBF to 2P-EXACT-COVER and 2P-EXACT-COVER to 2P-GEN-PARTITION) are given in [17]. They are repeated here for completeness.

Lemma 4.2 *2P-EXACT-COVER is PSPACE-complete.*

Proof: 2P-EXACT-COVER is obviously in $PSPACE$. Since the size of the game decreases, a simple exhaustive search with a stack can solve 2P-EXACT-COVER using a polynomial amount of space.

QBF can be viewed as a game between two players, called Exist and Forall. Given a formula $\varphi = \exists x_1 \forall x_2 \ldots \phi$, Exist begins play by choosing if x_1 is to be set to true of false. Forall then chooses if x_2 is to be set to true or false. Play continues with Exist and Forall alternating turns until all the variables have been given a truth assignment. Exists wins iff the truth assignment of the variables make ϕ true.

The basic idea is to convert an arbitrary formula $\varphi = \exists x_1 \forall x_2 \ldots \phi$ into an instance of 2P-EXACT-COVER such that Left's (Right's) decision over whether to choose X_i or $\overline{X}_i$ is equivalent to Exist's (Forall's) decision of whether to set x_i to be true or false. It will be shown that Left will be able to choose the Y_i subsets to form an exact cover iff ϕ is true under the specified truth assignments.

Without loss of generality, assume that the instance of QBF is in quantified 3SAT form, *i.e.*,

$$\varphi = \exists x_1 \forall x_2 \cdots \exists x_{2p} \forall x_{2p} \phi$$

[2]Note, in 2P-GEN-PARTITION, read as "two person generalized partition", B can be any value, and not necessarily equal to half of the sum of the given numbers.

$$\exists x_1 \forall x_2 \exists x_3 \forall x_4 (x_1 \vee x_2 \vee x_3)(\overline{x}_1 \vee \overline{x}_2 \vee x_4)(\overline{x}_3 \vee x_2 \vee \overline{x}_4) \Longrightarrow$$

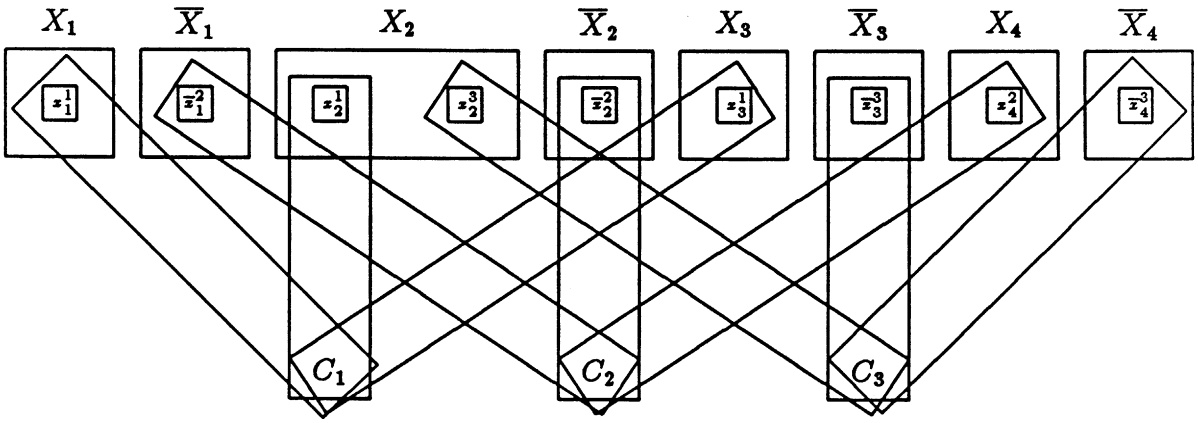


Figure 4.3: Transforming a Formula into an Instance of 2P-EXACT-COVER

where $\phi = C_1 \wedge \ldots \wedge C_n$ and each clause C_i is the disjunct of exactly three literals or negated literals. Morris transforms φ into an instance of 2P-EXACT-COVER via the following construction:

1. The grand set X contains the following $4m$ points:
   (a) One point, C_i, for each clause of ϕ.
   (b) One point, x_i^j, for each i,j such that x_i is a literal in clause C_j
   (c) One point, $\overline{x}_i^j$, for each i,j such that $\overline{x}_i$ is in clause C_j.
2. The subset X_i contains all points in X of the form x_i^j.
3. The subset $\overline{X}_i$ contains all points in X of the form $\overline{x}_i^j$.
4. The 6m Y_i subsets are defined as follows:
   (a) One subset for each point x_i^j in X.
   (b) One subset for each point $\overline{x}_i^j$ in X.
   (c) One subset for each pair of points (x_i^j, C_j) and $(\overline{x}_i^j, C_J)$ in X

For example, Figure 4.3 shows the transformation of a formula into an instance of 2P-EXACT-COVER.

There is a direct correspondence between play in an instance of QBF and play in the corresponding instance of a 2P-EXACT-COVER game. If a player sets a variable to false in QBF, then that variable can not be used to turn on any clause. This is analogous to a player in the 2P-EXACT-COVER game selecting the subset X_i, since it prevents Left from using any (x_i^j, C_j) subset to cover C_i. Similarly, if a player in QBF sets a variable to true, then that variable turns on any clause that contains it. This is analogous to a player in the EXACT-SET game selecting subset $\overline{X_i}$ and enabling any (x_i^j, C_j) to cover C_j.

Since the reduction can be done in polynomial time, and since the QBF game can be solved iff the 2P-EXACT-COVER can be solved 2P-EXACT-COVER is $PSPACE$-complete. ∎

Lemma 4.3 *2P-GEN-PARTITION is PSPACE-complete.*

Proof: 2P-GEN-PARTITION is obviously in $PSPACE$. Since the size of the game decreases, a simple exhaustive search with a stack can solve 2P-GEN-PARTITION using a polynomial amount of space.

Hence, it is sufficient to reduce 2P-EXACT-COVER to 2P-GEN-PARTITION. Let:

$$x_1, \overline{x}_1, \cdots x_n, \overline{x}_n, y_1, \cdots y_m$$

be the subsets of X in an instance of 2P-EXACT-COVER. To convert it into an instance of 2P-GEN-PARTITION is easy. If the set X contains k elements, then each subset is represented as a k digit, base four number. Each digit in the number represents an element in the set. The i^{th} digit of the number is set to one iff the subset contains the ith element of the set. Otherwise it is set to zero. B is set to $2^{k+1} - 1$, *i.e.*, 111...111.

Consider a typical solution to the instance of 2P-GEN-PARTITION constructed above. Since the sum of the selected numbers can never produce a carry, the only way to achieve B is to have, for all i, exactly one of the selected numbers have the i^{th} digit set to one. Hence, the solution represents an exact cover of X.

Since the reduction can be done in polynomial time, 2P-GEN-PARTITION is $PSPACE$-complete. ∎

Lemma 4.4 *2P-ALT is PSPACE-complete.*

Proof: 2P-ALT is obviously in $PSPACE$. Since the size of the game decreases, a simple exhaustive search with a stack can solve 2P-ALT using a polynomial amount of space.

Let the value b and the numbers:

$$x_0, \overline{x}_0, \cdots, x_n, \overline{x}_n, y_1, \cdots y_m$$

be an instance of 2P-GEN-PARTITION. Without loss of generality, assume that the largest value requires k bits to represent, and that n is odd. Then, the following instance of 2P-ALT is solvable iff the given instance of 2P-GEN-PARTITION is solvable:

$$X_0, \overline{X}_0, \cdots X_n, \overline{X}_n, Y_{11}, Y_{12}, \cdots, Y_{m1}, Y_{m2}$$

where

$$\begin{array}{lcl}
s & = & 2m + \log(m) + k \\
t & = & s + n \\
X_i & \rightarrow & \begin{cases} 2^{t-i} + x_i & \text{if i even} \\ 2^{t-i} - x_i & \text{if i odd} \end{cases} \\
\overline{X}_i & \rightarrow & \begin{cases} 2^{t-i} + \overline{x}_i & \text{if i even} \\ 2^{t-i} - \overline{x}_i & \text{if i odd} \end{cases} \\
Y_{j1} & \rightarrow & 2^{s-2i} + y_i \\
Y_{j2} & \rightarrow & 2^{s-2i} \\
B & \rightarrow & 2^{t-1} + 2^{t-3} + \cdots + 2^{s+1} + b
\end{array}$$

The high order bits of the X_i and $\overline{X}_i$ guarantee that their values are decreasing in size as i increases. This ensures in the alternating sum, the values Left selects will be added into the sum and that the values Right selects will be subtracted from the sum.

Ignoring the high order bits of X_i and $\overline{X}_i$, the effect of selecting x_i in 2P-GEN-PARTITION and the effect of selecting X_i in 2P-ALT is the same. If i is even, then in either case the value of x_i is added into the sum. If i is odd, then in the case of 2P-GEN-PARTITION, the value x_i is added into the sum. In the case of 2P-ALT, the value $-x_i$ is subtracted from the sum which is equivalent to adding x_i into the sum.

Unfortunately, the high order bits do have a secondary effect of adding into the sum $2^{t-1}+2^{t-3}+\cdots+2^{s+1}$. However, this extra amount is accounted for in the value of B.

The construction of the Y_{j1} and Y_{j2} is exactly the same as in the reduction of PARTITION to ALT. Again the construction guarantees that if Left selects Y_{j1} he must also select Y_{j2}. The net result of selecting Y_{j1} and Y_{j2} is that y_i is added to the sum.

Hence, there is a simple and direct correspondence between a solution to the given instance of 2P-GEN-PARTITION and 2P-ALT. x_i is in the solution of 2P-GEN-PARTITION iff X_i is in the solution of the corresponding instance of 2P-ALT. $\overline{x}_i$ is in the solution of 2P-GEN-PARTITION iff $\overline{X}_i$ is in the solution of the corresponding instance of 2P-ALT. y_j is in the solution of 2P-GEN-PARTITION iff Y_{j1} and Y_{j2} is in the solution of the corresponding instance of 2P-ALT.

Since the reduction can be done in polynomial time, 2P-ALT is $PSPACE$-complete. ∎

Corollary 4.2 *2P-SUBSET-SWITCH is PSPACE-complete.*

Proof: 2P-ALT and 2P-SUBSET-SWITCH are obviously equivalent since the value for Left when starting play in the sum of unbiased switches is the alternating sum of the component values of the switches.

Theorem 4.2 *$SUM_{d\leq 2}$ is PSPACE-complete*

Proof: $SUM_{d\leq 2}$ is obviously in $PSPACE$. Since the size of the game decreases, a simple exhaustive search with a stack can solve $SUM_{d\leq 2}$ using a polynomial amount of space.

Hence, if is sufficient to reduce 2P-SUBSET-SWITCH to $SUM_{d\leq 2}$. The difference between a 2P-SUBSET-SWITCH game and a SUBSET SWITCH game is that a 2P-SUBSET-SWITCH game contains an extra stage where Left and Right alternately select switches. Hence, it is sufficient to add components games to the sum constructed in Theorem 4.1 such that optimal play in the sum contains an extra stage where Left and Right alternately decide which switches are to remain in the game. In particular, transform

an arbitrary instance of 2P-SUBSET-SWITCH consisting of the value S and the switches:

$$\{ x_1 \mid -x_1 \}, \{ \overline{x}_1 \mid -\overline{x}_1 \}, \cdots, \{ x_{2n} \mid -x_{2n} \}, \{ \overline{x}_{2n} \mid -\overline{x}_{2n} \},$$
$$\{ y_1 \mid -y_1 \}, \cdots, \{ y_m \mid -y_m \}$$

into an instance of $SUM_{d \leq 2}$, construct the game:

$$J + K_{11} + K_{12} + \cdots + K_{n1} + K_{n2} + G_{11} + G_{12} + \cdots + G_{m1} + G_{m2} + H + I + \uparrow$$

where

$$\begin{array}{lcl}
J & \rightarrow & \{ \{ \infty \mid \sum_{i=1}^{n} A_i - C \} \mid -\infty \} \\
K_{i1} & \rightarrow & \{ \{ x_{2i-1} \mid -x_{2i-1} \}, \{ \overline{x}_{2i-1} \mid -\overline{x}_{2i-1} \} \mid -F_{2i-1} \} \\
K_{i2} & \rightarrow & \{ F_{2i} \mid \{ x_{2i} \mid -x_{2i} \}, \{ \overline{x}_{2i} \mid -\overline{x}_{2i} \} \} \\
G_{j1} & \rightarrow & \{ \{ y_i \mid -y_i \} \mid -A_j \} \\
G_{j2} & \rightarrow & \{ 0 \mid -A_j \} \\
H & \rightarrow & \{ C \mid -C \} \\
I & \rightarrow & \{ D \mid -S, \{ S \mid -E \} \} \\
\uparrow & \rightarrow & \{ 0 \mid \{ 0 \mid 0 \} \}
\end{array}$$

and

$$\begin{array}{lcl}
F_i & > & 4 * F_{i+1} \\
F_n & > & 4 * A_1 \\
A_i & > & 4 * A_{i+1} \\
A_n & > & 4 * (C + D + E + S + \sum_{i=1}^{n} x_i) \\
C & > & D + \sum_{i=1}^{n} x_i \\
D & > & 4 * E \\
E & > & 2 * \sum_{i=1}^{n} x_i .
\end{array}$$

The only significant difference between this game and the game constructed for Theorem 4.1 is that this game contains a number of K component games. Optimal play in both games is very similar.

As in the game constructed for Theorem 4.1, optimal play begins with Left moving in J and Right responding taking $\sum_{i=1}^{n} A_i - C$. If Left does not move in J, Right can gain $-\infty$ and win. Similarly, after Left plays in J, he is threatening to gain ∞. Right must respond in the same game to prevent this threat.

Next, Left must play in K_{11}. If he does not, Right will pick up $-F_1$. Since F_1 is greater than the sum of all other values in the sum, Left is better off playing in K_{11} then playing first in all other games.

Similarly, Right then must play in K_{12}. If he does not, Left will pick up F_2. Since F_2 is greater than the sum of all other values in the sum, Right could play first in all other games and still not recoup her loss.

By Repeating the above argument, it is easy to see that the K games will be played out in order. Left and Right will alternately make the decision of whether to include $\{ x_i \mid -x_i \}$ or $\{ \overline{x}_i \mid -\overline{x}_i \}$ into the rest of the game.

After the K_i games have been played out, the game is exactly like the game constructed in Theorem 4.1 except that the game contains an additional $2n$ switches. However, the additional switches do not affect optimal play as the games G_{j1}, G_{j2}, H, and I are played out.

So, as in the game constructed in Theorem 4.1, play then proceeds in the G games. Left begins by playing in either G_{11} or G_{12}, and hence deciding if the switch $\{y_1 \mid -y_1\}$ should be selected or not. Right responds in the other game. Play proceeds down the pairs of G_i's, with Left choosing whether or not to select each $\{ y_i \mid -y_i \}$.

After the G game is played out, Left must play in H and Right gains control over the play.

Let L be V_L of the remaining switches. Right must decide which option of I to play in. If he plays in first option, Left will start the play in the remaining switches. The switches will be played out and the final score will be $-S+L$. If he plays in the second option, Left will respond in $\{S \mid -E\}$ taking S and Right will start the play in the switches. The final score will be $S-L$.

Thus, Left will win iff both of Right's options (taking $S-L$ or $L-S$) result in a non negative final score. This is true iff Left solved the 2P-SUB-SET-SWITCH problem.

Since the reduction can be done in polynomial time, $SUM_{d\leq 2}$ is $PSPACE$-complete. ∎

Chapter 5

Coping with SUM

Since SUM is $PSPACE$-complete, it is unlikely that an efficient algorithm for solving it will be discovered. Two alternate approaches exist for coping with an instance of SUM. The first is to relax the criteria of success. Instead of requiring optimal solutions, an algorithm is only required to produce a solution that is close to optimal. The second alternative is to do an exponential time search for the optimal solution.

Heuristic solutions are obviously beneficial when speed is critical and small errors in the solution are tolerable.

A "good" though not optimal solution to SUM can have one of two forms. It can approximate the value of the final score when both players play optimally, or it can heuristically choose a move. This thesis focuses on algorithms that approximate the final score to within a known error.

If $P \neq PSPACE$, then there are constraints on how good any polynomial time approximation can be. Assume that an efficient heuristic solution approximates the value of the final score under optimal play to within some specified accuracy. By a simple "change of currency" argument, it is easy to show that the accuracy can not be within a constant of the optimal solution. Similarly, the accuracy can not be within a multiple of the optimal solution since determining if the final score of an instance of SUM equals zero is $PSPACE$-complete.

However, Hanner[7] proved that it is possible to approximate the final score of a sum of games to within an accuracy dependent solely upon the "worst" component game. His bounds are independent on the complexity of the component games, and on the number of component games. In

particular, he proved that in the sum $G_1 + \ldots + G_n$, it is possible to approximate the final score to within $max\{\sigma(G_i) \mid 1 \leq i \leq n\}$ of the optimum solution.

This chapter presents a revised version of Hanner's proof. Hanner's result was obtained by imagining that each player in a game is forced to pay a *tax* for the privilege of moving. The fact that this yields results which apply to normal play seems to be a bit magical. This chapter recasts the work in a more intuitive light, and develops the mathematical notation required for a clean and concise proof.

This chapter also improves upon Hanner's result to show that the final score of $G_1 + \ldots + G_n$ can be approximated to within the *second largest* $\sigma(G_i)$.

The second approach to handling an instance of SUM is to perform a min-max search for the optimal solution. Such algorithms require exponential time. However, their running time can be greatly improved upon by pruning techniques. This chapter considers how the approximate solutions developed in the chapter can be used to guide and prune the search for an optimal solution. Particular attention will be paid to the use of the approximate algorithms in conjunction with Berliner's B^*[4] search algorithm.

One consequence of using B^* is that it increases the motivation for finding better approximate solutions. The better the approximate solution, the more "informed" and hence the better the B^* search will be.

5.1 Heuristic Solutions

This section places bounds on the final score of a sum of games. The basic approach is to define a heuristic strategy, and then show that if Left uses the strategy he can force the final score be at least some minimum value, and if Right uses the strategy he can hold down the final score to be at most some maximum value. Hence, the true value of the game is guaranteed to be between the minimum and maximum value.

This section outlines four different results. The first, by Milnor[16], is obtained via a *follow the leader* strategy. The second, by Hanner[7], is obtained via the *mean* strategy. The last two results are improvements on Hanner's.

5.1.1 Follow the Leader Strategy

The basic strategy used for approximating the value of a game is the following *follow the leader* strategy:

1. Only play locally optimal moves.
2. Whenever possible, play in the same game as the opponent.

The purpose of the strategy is to simplify the play in a sum of games. Each component game is played out as if it were the sole game in the sum. Thus, it is possible to approximate the final score for G in terms of the its component games.

The *follow the leader* strategy was first used by Milnor[16][1] to bound the sum of two games. If $G = G_1 + \ldots + G_n$, then the generalized version of the result is:

$$V_L(G_1) + V_R(G_2) + \cdots + V_R(G_n) \leq V_L(G) \leq V_L(G_1) + \cdots + V_L(G_n).$$

To prove the right hand side of the inequality, assume that Right adopts the *follow the leader* strategy: If Right always plays in the same game as Left, then each component game G_i will be played out as if it was the only game in the sum. Furthermore, if both Left and Right play locally optimal moves, then the final score of each component game will be $V_L(G_i)$. Thus, the final score of the sum will be

$$V_L(G_1) + \ldots + V_L(G_n).$$

Left can not increase this final score. If Left plays any move that is not locally optimal, the final score will be the same or lowered.

If in one of the games Left plays into an end position, Right will be forced to play twice (or first) in some other game. Since its always to Right's advantage to play twice in a game (or to start a game), this only serves to lower the final score. Hence by using the strategy Right can guarantee that

$$V_L(G_1 + \cdots + G_n) \leq V_L(G_1) + \cdots + V_L(G_n)$$

[1]Milnor's and Hanner's model of a game is slightly different than Conway's. They do not make a clean distinguish between a number and a game. However, their results hold in Conway's model.

Similarly, to prove the left hand side of the inequality, assume that Left follows Right and responds with locally optimal moves. Also assume that Left's first move is the locally optimal move in G_1 to G_1^L. If Left is always able to play in the same game as Right, and if Right always makes locally optimal moves than the final score will be

$$V_R(G_1^L) + V_R(G_2) + \cdots + V_R(G_n).$$

The situation can only improve for Left if Right does not make locally optimal moves, or if Right plays into an end position forcing Left to play twice (or first) in the another game. Since $V_R(G_1^L) = V_L(G_1)$, this yields:

$$V_L(G_1) + V_R(G_2) + \cdots + V_R(G_n) \leq V_L(G_1 + \cdots + G_n) \blacksquare$$

5.1.2 Mean Strategy

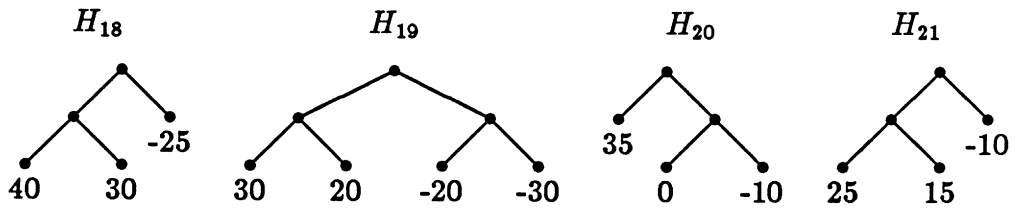
The *follow the leader* strategy, as used by Milnor, is not very powerful. The game is played out in a way that strongly favors the "leader". Hanner[7] improves on Milnor's result by devising a more powerful *follow the leader* strategy. In particular, he defines the *mean* strategy which is a *follow the leader* strategy where the follower always plays *t-optimal* moves. Using the strategy, he proves that if $G = G_1 + \ldots + G_n$ then

$$Mean(G) \leq V_L(G) \leq Mean(G) + Max(\sigma(G_i))$$

Hanner's bounds are surprisingly accurate. They are dependent solely on the hottest game. Thus, unlike Milnor's bounds, the accuracy of the bounds do not decrease as the number of games in the sum increases. Any number of cooler games can be included in the sum, and the accuracy remains the same. For example, Figure 5.1 compares the accuracy of Hanner's and Milnor's bounds.

Before defining t-optimal moves, it is necessary to generalize the notation for ${}^tL_n(G)$, ${}^tR_n(G)$, ${}^tV_L(G)$, and ${}^tV_R(G)$ to include the concept of a player using a specific strategy. In particular:

${}^tL_n^{xy}(G) \Rightarrow$ The position that results after n moves when a tax of t is imposed on each move, both players play alternately, Left starts, Left plays according to strategy x, and Right plays according to strategy y. If no strategy is specified, then an optimal strategy is assumed.



V_L		30	20	35	15
V_R		-25	-20	0	-10
Mean:		5	0	15	5
σ		30	20	20	15

		$H_{18}+H_{19}$	$H_{18}+H_{19}+H_{20}$	$H_{18}+H_{19}+H_{20}+H_{21}$
Milnor:	bounds	$10 \le V_L \le 50$	$10 \le V_L \le 85$	$0 \le V_L \le 100$
	accuracy	40	75	100
Hanner:	bounds	$5 \le V_L \le 35$	$20 \le V_L \le 50$	$25 \le V_L \le 65$
	accuracy	30	30	30

Figure 5.1: Comparing the Accuracy of Hanner's and Milnor's Results

${}^tR_n^{xy}(G) \Rightarrow$ has the same meaning as ${}^tL_n^{xy}(G)$ except that Right starts the play in G.

${}^tV_L^{xy}(G) \Rightarrow$ is the final score when a tax of t is imposed on each move, both players play alternately, Left starts, Left plays according to strategy x, and Right plays according to strategy y. If no strategy is specified, then an optimal strategy is assumed.

${}^tV_R^{xy}(G) \Rightarrow$ is the same as ${}^tV_L^{xy}(G)$ except that Right starts the play in G.

Furthermore, the following notation is used for referring to particular strategies:

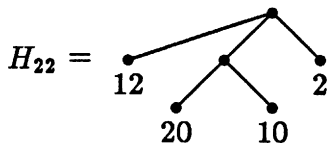
- "0" $\rightarrow$ optimal strategy
- "-" $\rightarrow$ arbitrary strategy
- "t" $\rightarrow$ t-optimal strategy (to be defined).

For example, $R_5^{-0}G$ is the position reached after 5 moves in G when Left plays an arbitrary strategy, Right plays optimally, and Right plays first. ${}^tV_L^{0-}(R_5^{-0}G)$ is the final score when $R_5^{-0}G$ is played such that Left plays only optimal moves, Right plays an arbitrary strategy, Left starts the play, and a tax of t is applied to each move.

In the notation for ${}^tL_n^{xy}(G)$, ${}^tR_n^{xy}(G)$, ${}^tV_L^{xy}(G)$, and ${}^tV_R^{xy}(G)$, the strategy x and y is defined after the rules of the game are defined, *i.e.*, the strategy is dependent upon whether a tax of t is imposed on every move. For example, in general, ${}^tL_1^{0-}(G) \neq L_1^{0-}(G)$ since the optimal move in a taxed game is different from the optimal move in a untaxed game.

A t-optimal move in a (untaxed) game is defined to be the move that *would be* optimal if tax of t was imposed on every move in the game. However, t-optimal moves only exist for $t \leq \sigma(G)$. If a tax of $t > \sigma(G)$ is imposed on every move in G, then it is in neither player's advantage to move first and the game is a number. Hence, the t-optimal move in the untaxed version of the game is undefined. Thus, a t-optimal for Left is defined as follows:

$$L_1^{t-}(G) = \begin{cases} {}^tL_1^{0-}(G) & \text{if } t \leq \sigma(G) \\ \text{undefined} & \text{if } t > \sigma(G) \end{cases} \tag{5.1}$$



$$\sigma(H_{22}) = 8$$

$$\begin{array}{llll} L_1^{0-}H_{22} = 12 & & R_1^{0-}(H_{22}) = 2 \\ L_1^{1-}H_{22} = 12 & & R_1^{1-}(H_{22}) = 2 \\ L_1^{3-}H_{22} = \{\,20 \mid 10\,\} & & R_1^{3-}(H_{22}) = 2 \\ L_1^{9-}H_{22} = \text{undefined} & & R_1^{9-}(H_{22}) = \text{undefined} \end{array}$$

Figure 5.2: t-optimal moves in H_{22}

The "t" in "t-optimal" is a variable that can take on different values. For example, Figure 5.2 shows the t-optimal moves in H_{22} for various values of t. Note, a 0-optimal move is equivalent to an optimal move since a tax of zero is equivalent to no tax.

Hanner provides some motivation for using a strategy with t-optimal moves by stating that

> When a player shall move in a sum of games he chooses one game, say G, and there makes a move. Thereby he loses the possibility to make the move in one of the other games. If the value of this possibility is put equal to t it is natural to compare the situation with the case when the player has to move in G and pay the amount t to the other player when moving.

Unfortunately, Hanner does not expound any further on the nature of t-optimal moves. The important question of how t-optimal moves work remains.

One way to view Hanner's strategy, is that it addresses a basic weakness found in Milnor's strategy, *i.e.*, tempo. In Milnor's strategy, the follower is a wimp that passively responds to the leader's move, even when it is obvious that the leader's move is not *sente*. For example, consider the game shown in Figure 5.3. The first game is *hot*, and it is natural to assume that the

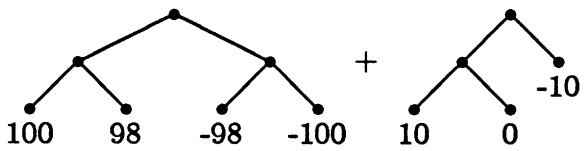


Figure 5.3: A Sum Where the First Move Is Not Sente

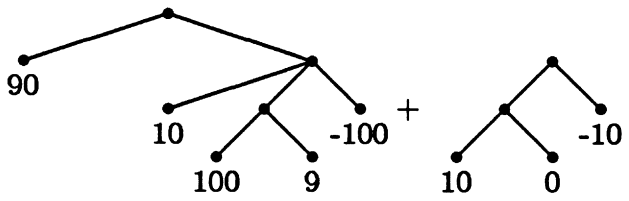


Figure 5.4: A Sum Where the Locally Optimal Response Is Incorrect

leader will play there. However, the follower gains very little by responding to the leader's move.

In Milnor's strategy, the follower's ignorance of tempo is also illustrated by his exclusive use of locally optimal moves. A non-optimal move that forces a response from one's opponent is often better than a locally optimal move that can be ignored. For example, consider the game shown in Figure 5.4. Assume that the leader (Right) played in the first game. The locally optimal response it to reduce the game to 10. However, from a global perspective, it is better for the follower to reduce the game to $\{ 100 \mid 9 \}$, force the leader to take 9, and then play first in the other component.

Hanner's strategy addresses the issue of tempo via *taxation*. Taxation provides some insight into the value of a move. It was used in section 2.6 to efficiently compute the temperature of a game. Here, in the form of t-optimal moves, it is used to provide the follower with some intuition about tempo. In particular, before using Hanner's *mean* strategy it is necessary

to set the value of t. This sets a minimum threshold of t on the value of a move. The effect is two fold.

First, t-optimal moves only exist if a move is worth at least t. Hence, the follower can only respond in the same game as the leader when doing so is worth at least t. For example, reconsider Figure 5.3. A t-optimal response to the leader's move in the first game exists only when $t \leq 1$. That is, the follower can only respond in the first game if the threshold is set to be less than or equal to one.

Second, setting the value of t sets minimum threshold on the amount the follower can lose in a local situation in order to keep sente. For example, reconsider Figure 5.4. After the leader (Right) plays in the first game, the follower(Left) has two options. Under Hanner's *mean* strategy, which option is chosen depends upon the value of t. If $t \geq 1$ then the t-optimal response is to move to $\{\, 100 \mid 9 \,\}$. That is, Left takes a local loss of 1 (he gets 9 instead of 10) in order to move first in the other component game. On the other hand, if $t < 1$, then the follower is unwilling to take a loss of 1. The t-optimal response is to take 10.

What follows is a revised version of Hanner's proof. The proof is divided into two parts. The first part proves that in a single game G_i, a player playing $\sigma(G_i)$-optimal moves can force the mean of final position of G_i to be within $\sigma(G_i)$ of $mean(G_i)$. The second part proves that in a sum of game $G = G_1 + \ldots + G_n$, a player has a strategy that forces the final score to be close to its mean value.

Lemma 5.1 *Assume $t \geq \sigma(G)$. Consider a sequence of n moves in G such that Left always has a t-optimal move, i.e.,*

$$\forall i,\ 1 \leq i \leq n-1,\ \sigma(L_i^{t^-}(G)) > t.$$

If Left always plays a t-optimal move, then the following equations describe how $mean(G)$ compares to the mean of the final position in the sequence based upon who moves first and last in the sequence.

$$\begin{array}{lcll} Mean(L_{2k+1}^{t^-}G) & \geq & Mean(G) + t & \\ Mean(L_{2k}^{t^-}G) & \geq & Mean(G) & \text{if } \sigma(L_{2k}^{t^-}G) \leq t \\ Mean(R_{2k}^{t^-}(G)) & \geq & Mean(G) & \\ Mean(R_{2k+1}^{t^-}(G)) & \geq & Mean(G) - t & \text{if } \sigma(R_{2k+1}^{t^-}G) < t \end{array}$$

In general, it is hard to prove anything about non-optimal moves in a game. However, a t-optimal move in a untaxed game is the optimal move in a taxed game. Hence, questions about t-optimal moves in an untaxed game can be answered by considering optimal moves in a taxed game.

In particular, consider a sequence of moves in a taxed game. Assume both players play optimally. If there are an even number of moves, then the taxes Left pays to Right and the taxes Right pays to Left will cancel out. If there are an odd number of moves, then one player will pay an extra tax. By repeatedly applying equations 2.3 and its dual, this can be stated algebraically as follows:

$$\begin{aligned}
{}^tV_R({}^tL_{2k+1}(G)) &= {}^tV_L(G) + t \\
{}^tV_L({}^tL_{2k}(G)) &= {}^tV_L(G) \\
{}^tV_R({}^tR_{2k}(G)) &= {}^tV_R(G) \\
{}^tV_L({}^tR_{2k+1}(G)) &= {}^tV_R(G) - t
\end{aligned}$$

This is the heart of the lemma. The rest of the proof simply massages the above equations into the desired form.

The left hand side of the above equations assume that both players play optimally. However, if Right does not play optimally this only helps Left. So:

$$\begin{aligned}
{}^tV_R{}^tL^{0-}_{2k+1}(G) &\geq {}^tV_R{}^tL_{2k+1}(G) &= {}^tV_L(G) + t \\
{}^tV_L{}^tL^{0-}_{2k}(G) &\geq {}^tV_L{}^tL_{2k}(G) &= {}^tV_L(G) \\
{}^tV_R{}^tR^{0-}_{2k}(G) &\geq {}^tV_R{}^tR_{2k}(G) &= {}^tV_R(G) \\
{}^tV_L{}^tR^{0-}_{2k+1}(G) &\geq {}^tV_L{}^tR_{2k+1}(G) &= {}^tV_R(G) - t
\end{aligned}$$

By equation 5.1 optimal moves in a taxed game are t-optimal moves in a non-taxed game. So,

$$\begin{aligned}
{}^tV_RL^{t-}_{2k+1}(G) &= {}^tV_R{}^tL^{0-}_{2k+1}(G) &\geq {}^tV_L(G) + t \\
{}^tV_LL^{t-}_{2k}(G) &= {}^tV_L{}^tL^{0-}_{2k}(G) &\geq {}^tV_L(G) \\
{}^tV_RR^{t-}_{2k}(G) &= {}^tV_R{}^tR^{0-}_{2k}(G) &\geq {}^tV_R(G) \\
{}^tV_LR^{t-}_{2k+1}(G) &= {}^tV_L{}^tR^{0-}_{2k+1}(G) &\geq {}^tV_R(G) - t
\end{aligned}$$

The right hand side of all the above equations can be simplified since by Equation 2.6 when $t \geq \sigma(G), {}^tV_R(G) = {}^tV_R(G) = Mean(G)$. So,

$$\begin{aligned}
{}^tV_RL^{t-}_{2k+1}(G) &\geq {}^tV_L(G) + t &= Mean(G) + t \\
{}^tV_LL^{t-}_{2k}(G) &\geq {}^tV_L(G) &= Mean(G) \\
{}^tV_RR^{t-}_{2k}(G) &\geq {}^tV_R(G) &= Mean(G) \\
{}^tV_LR^{t-}_{2k+1}(G) &\geq {}^tV_R(G) - t &= Mean(G) - t
\end{aligned}$$

Finally, the left hand side of the first and third equations can be simplified since the mean value of a position is always greater than the value for Right (equation 2.6 combined with equation 2.4). The second and fourth equations can be simplified using equations 2.6 assuming that σ of the final position in the sequence is lower than t. So,

$$\begin{array}{lcll} Mean(L^{t-}_{2k+1}G) & \geq & {}^tV_R L^{t-}_{2k+1}(G) & = Mean(G)+t \\ Mean(L^{t-}_{2k}G) & \geq & {}^tV_L L^{t-}_{2k}(G) & = Mean(G) \quad \text{if } \sigma(L^{t-}_{2k}G) \leq t \\ Mean(R^{t-}_{2k}G) & \geq & {}^tV_R R^{t-}_{2k}(G) & = Mean(G) \\ Mean(R^{t-}_{2k+1}G) & \geq & {}^tV_L R^{t-}_{2k+1}(G) & = Mean(G)-t \quad \text{if } \sigma(R^{t-}_{2k+1}G) \leq t \end{array} \blacksquare$$

Theorem 5.1 *For the game $G = G_1 + \ldots + G_n$, let:*

$$Mean(G) = Mean(G_1 + \cdots + G_n)$$

$$\sigma = max\{\sigma(G_k) \mid 1 \leq k \leq n\}$$

If Left is the first player to move, then:

$$Mean(G) \leq V_L(G_1 + \cdots + G_n) \leq Mean(G) + \sigma$$

Proof: The proof proceeds by induction on the number of moves in a game. Let $l(G)$ be the maximum number of of moves that can be played in G.

Basis: If $l(G_1 + \ldots + G_n) = 0$ then all G_k, $1 \leq k \leq n$, are end positions, *i.e.*, numbers. Thus, $\sigma(G_k) = 0$ and $mean(G_k) = V_L(G_k)$. The theorem holds.

Induction Step: Assume the theorem holds for all games such that $l(G_1 + \ldots + G_n) \leq m$. It will be shown that it holds for all games such that $l(G_1 + \ldots + G_n) = m + 1$.

The proof will center around the *mean* strategy. The *mean* strategy is the *follow the leader* strategy described below:

1. Always play in the same game as your opponent except if you must make the first move.
2. Only play σ-optimal moves.

It will not always be possible to play according to the above strategy since σ-optimal moves do not always exist. In particular, the strategy is considered valid only until one of the following two conditions occur:

α : The opponent plays in G_i leaving position P_i such that $\sigma(P_i) \leq \sigma$

β : Positions $P_r, 1 \leq r \leq n$ have been reached for which $\sigma(P_r) \leq \sigma$.

To prove the left hand side of the inequality, it is sufficient to show that if Left uses the *mean* strategy until one of the two stopping conditions occur, then G will be reduced to a game $P = P_1 + \ldots + P_n$ such that if its Left's turn to play in P then $V_L(P) \geq Mean(G)$ where as if its Right's turn to play in P then $V_R(P) \geq Mean(G)$.

Consider how the mean of each game P_i compares to the mean of the corresponding G_i game. Each P_i will have one of the following four forms $L^{\sigma-}_{2k+1}G_i$, $L^{\sigma-}_{2k}G_i$, $R^{\sigma-}_{2k+1}G_i$, or $R^{\sigma-}_{2k}G_i$. Lemma 5.1 specifies to for each i : $1 \leq i \leq n$:

$$\begin{array}{lcll} Mean(L^{\sigma-}_{2k+1}G_i) & \geq & Mean(G_i) + t & \\ Mean(L^{\sigma-}_{2k}G_i) & \geq & Mean(G_i) & \text{if } \sigma(L^{\sigma-}_{2k}G_i) \leq t \\ Mean(R^{\sigma-}_{2k}G_i) & \geq & Mean(G_i) & \\ Mean(R^{\sigma-}_{2k+1}G_i) & \geq & Mean(G_i) - t & \text{if } \sigma(R^{\sigma-}_{2k+1}G_i) \leq t \end{array}$$

It is important to note that if Right makes the last move in the ith component game reducing it to P_i, then it must be the case that $\sigma(P_i)$ is greater than σ (otherwise Left would have responded with a σ-optimal move). Hence, the conditions applying to the second and fourth equations above are satisfied.

The above four formulas can be condensed into the following equation:

$$Mean(P_i) \geq Mean(G_i) + l_{L,i}\sigma - l_{R,i}\sigma$$

where $l_{L,i}$ is the number of moves made by Left in G_i and $l_{R,i}$ is the number of moves made by Right in G_i. Taking the sum of the inequalities for all $i, 1 \leq i \leq n$ produces

$$Mean(P) \geq Mean(G) + l_L\sigma - l_R\sigma \qquad (5.2)$$

where l_L and l_R are the number of moves made by Left and Right respectively.

Consider the number of moves made when G is reduced to P. If it is even, then $l_L = l_R$ and it is Left's turn to move in P. By the induction hypothesis and equation 5.2

$$V_L(P) \geq Mean(P) \geq Mean(G)$$

If the number of moves is odd, then $l_L = l_R + 1$ and its Right's turn to move in P. Play must have ended due to the β stopping condition. Hence, $max\{\sigma(P_i)|1 \leq i \leq n\} \leq \sigma$. This, combined with the dual of the induction hypothesis and equation 5.2 yields:

$$\begin{aligned} V_R(P) &\geq Mean(P) - max\{\sigma(P_i)|1 \leq i \leq n\} \\ &\geq Mean(P) - \sigma \\ &\geq Mean(G) + \sigma - \sigma \\ &\geq Mean(G) \end{aligned}$$

Thus, by playing σ-optimal moves Left can reduce G to some game P such that the value of P is greater than or equal to the mean value of G. The left hand side of the inequality is proven.

To prove the right hand side of the inequality, it is sufficient to show if Right uses the follow-the-leader strategy, then G will be reduced to a game $P = P_1 + \ldots + P_n$ and the value of P is less than $Mean(G) + \sigma$.

Consider a component games P_i of P. By construction Right will never play first in a game. Hence, the dual of the first two equations of Lemma 5.1 specify how the mean of each P_i compares to the mean value of the corresponding G_i. That is, for each $i, 1 \leq i \leq n$:

$$\begin{aligned} Mean(L_{2k}^{-\sigma}G) &\leq Mean(G_i) \\ Mean(L_{2k+1}^{-\sigma}G) &\leq Mean(G_i) + \sigma \end{aligned}$$

Proceeding as before, the above equations are reduced to

$$Mean(P_i) < Mean(G_i) + l_{L,i}\sigma - l_{R,i}.$$

Summing the above equation for all i yields:

$$Mean(P) < Mean(G) + l_L\sigma - l_R\sigma \tag{5.3}$$

Consider the number of moves made when G is reduced to P. If it is even, then $l_L = l_R$ and it is Left's move in P. Play must have ended due to the β stopping condition. Hence, $max\{\sigma(P_i)|1 \leq i \leq n\} \leq \sigma$. This, combined with the dual of the induction hypothesis and equation 5.3 yields:

$$\begin{aligned} V_L(P) &\leq Mean(P) + max\{\sigma(P_i) \mid 1 \leq i \leq n\} \\ &\leq Mean(P) + \sigma \\ &\leq Mean(G) + \sigma \end{aligned}$$

If the number of moves is odd, then $l_L = l_R + 1$ its Right's move in P. By the dual of the induction hypothesis and equation 5.3:

$$V_R(P) \leq Mean(P) \leq Mean(G) + \sigma.$$

Thus, the right hand side of the inequality is proven. ∎

5.1.3 Improving Hanner's Bounds

Hanner's strategy is actually better than Hanner claimed. Consider what happens to Hanner's bounds as G' in Figure 5.5 is played out. Initially, Hanner's bounds are quite accurate, *i.e.*, if Left plays first in G' then the final score will be between -90 and -64. After Left moves the game becomes more volatile and the accuracy of Hanner's bounds decrease, *i.e.*, $-110 \leq V_L(G) \leq 10$.

However, the bounds computed for G' also apply to G. Consider the upper bound on G'. Hanner proves that no matter how Left plays, Right has a strategy that guarantees to hold the final score below -64. In particular, if Left moves to G, Hanner proves that Right has a strategy that guarantees the final score will be less than or equal to -64, Therefore, Right, starting from G, has a strategy that holds the final score down to -64 and it is fair to claim that $V_R(G) \leq -64$.

This suggests an interesting way to improve on Hanner's bounds. Assume that the game originally began with Right to move first in G. Hanner's bounds would not be very informative. Better bounds could be computed simply by *imagining* that the game actually began with G' and Left to move.

What follows is a proof that the value of a game, $G = G_1 + \ldots + G_n$ can be estimated to within the *second largest* $\sigma(G_i)$. The basic idea is that a

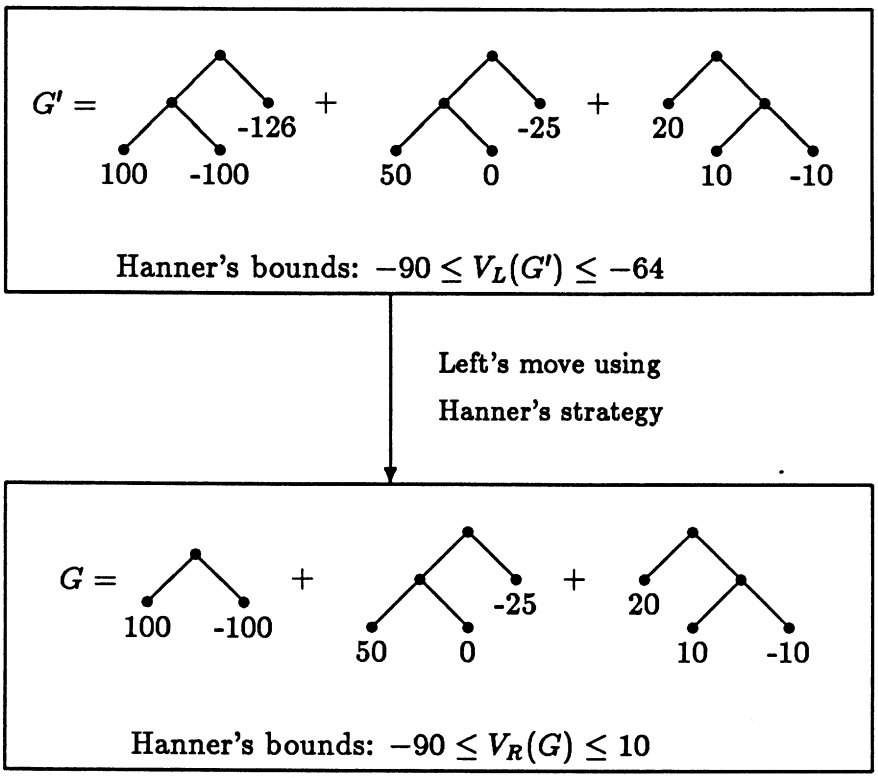


Figure 5.5: Hanner's Bounds as G' Is Played

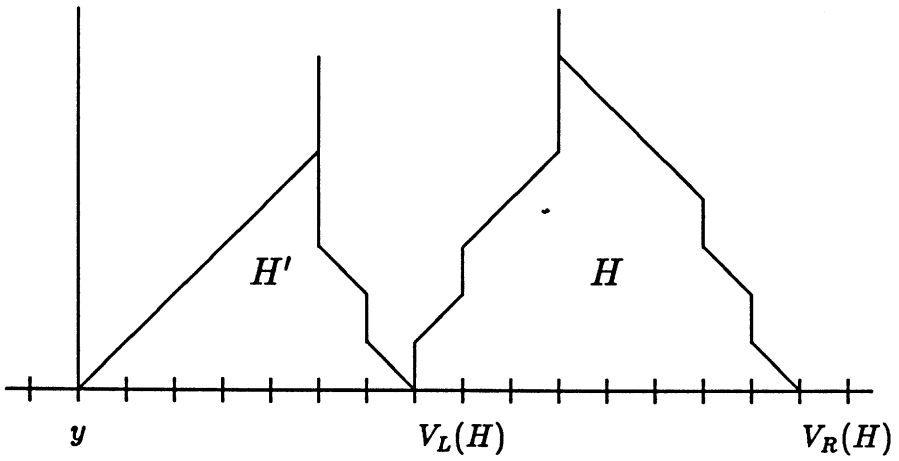


Figure 5.6: Constructing the Thermograph for H'

new game G' is constructed such that the $\sigma(G')$ equals the second largest $\sigma(G_i)$ and in a way that guarantees that Hanner's bounds on G' also apply to G. The ability to construct such a game is proven in the following two lemmas:

Lemma 5.2 *For any game H and number $x \geq 0$, a new game H' can be constructed such that $\sigma(H') = x$ and $\forall s, L_1^{s-}(H') = H$, i.e., the only Left option from H' is to move to H.*

Proof: Let H' be $\{ y \mid H \}$ for some value of y. Consider an arbitrary thermograph for H and the resulting thermograph for H' as shown in Figure 5.6. It is obvious that one can always set the value of y such that $\sigma(H')$ is as high or as low as one wants. ∎

Lemma 5.3 *Let $G = G_1 + \ldots + G_n$ and $\sigma = max\{\sigma(G_i) \mid 1 \leq i \leq n\}$. Then,*

$$Mean(G) \leq V_R(L_1^{\sigma-}G) \leq Mean(G) + \sigma.$$

Proof: The basic idea is that Hanner's bounds on G also apply to the game that results when Left plays a σ-optimal move.

Hanner proves that if Left moves to $L_1^{\sigma-}G$ then he can force the final score to be greater than or equal to $Mean(G)$, *i.e.*,

$$Mean(G) \leq V_R(L_1^{\sigma-}G).$$

Similarly, Hanner proves that for any Left move in G, Right can respond such that the final score is less than $Mean(G) + \sigma$. In particular, if Left moves to $L_1^{\sigma-}G$, Right can respond such that the final score is less than $Mean(G) + \sigma$. Hence,

$$V_R(L_1^{\sigma-}G) \leq Mean(G) + \sigma. \blacksquare$$

Theorem 5.2 *Let $G = G_1 + G_2 + \ldots + G_n$ where $\forall i, 1 \leq i \leq n-1, \sigma(G_i) \geq \sigma(G_{i+1})$. Then, it is possible to approximate the value of $G_1 + \ldots + G_n$ to within $\sigma(G_2)$ of the optimal value.*

Proof: Construct a new game H_1 out of G_1 such that $\forall s, L_1^{s-}H_1 = G_1$ and $\sigma(H_1) = \sigma(G_2)$. Lemma 5.2 guarantees that this can be done.

Let $G' = H_1 + G_2 + \ldots + G_n$. Lemma 5.3 guarantees

$$Mean(G') \leq V_R(L_1^{\sigma-}G') \leq Mean(G') + \sigma(G').$$

One σ-optimal move in G' is to move in H_1. This reduces the G' to G. So,

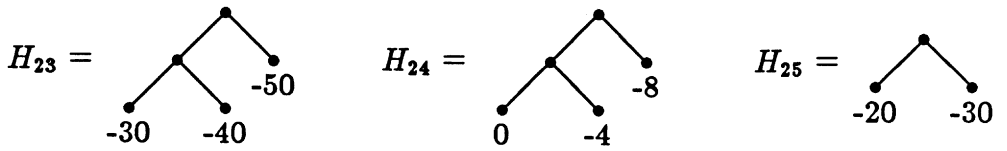
$$Mean(G') \leq V_R G \leq Mean(G') + \sigma(G').$$

Furthermore:

$$\sigma(G') = max(\{\sigma(G_i) \mid 1 \leq i \leq n\}) = \sigma(G_1) = \sigma(G_2)$$

Hence,

$$Mean(G') \leq V_R G \leq Mean(G') + \sigma(G_2). \blacksquare$$



$Mean(H_{23}) = -42.5$
$\sigma(H_{23}) = 7.5$
${}^5V_L H_{23} = -40$
${}^5V_R H_{23} = -45$

$Mean(H_{24}) = -5$
$\sigma(H_{24}) = 3$
${}^5V_L H_{24} = -5$
${}^5V_R H_{24} = -5$

$Mean(H_{25}) = -25$
$\sigma(H_{25}) = 5$
${}^5V_L H_{25} = -25$
${}^5V_R H_{25} = -25$

Figure 5.7: Defining H_{23}, H_{24}, and H_{25}

5.1.4 Thermostatic Strategy

Berlekamp, Conway, and Guy [2] also improved on the mean strategy. In particular, if $G = G_1 + \ldots + G_n$, then for all t,

$$\begin{aligned} {}^tV_L(G_1) + {}^tV_R(G_2) + \cdots + {}^tV_R(G_n) &\leq {}^tV_L(G) \\ &\leq {}^tV_L(G_1) + \cdots + {}^tV_L(G_n) + t \end{aligned}$$

This is a generalization of Hanner's and Milnor's results. If $t = 0$, then the result is identical to Milnor's. If $t = \sigma(G)$ then the result is identical to Hanner's.

For example, consider the three games, H_{23}, H_{24}, and H_{25}, shown in Figure 5.7. By Hanner's result (Theorem 5.1):

$$-72.5 \leq V_L(H_{23} + H_{24} + H_{25}) \leq -65$$

However, if $t = 5$ then Berlekamp, Conway, and Guy prove that:

$$-70 \leq V_L(H_{23} + H_{24} + H_{25}) \leq -65$$

Berlekamp, Conway, and Guy[2] provide an efficient algorithm for finding the t that produces the tightest bounds based on *thermographs* and they give an argument of the correctness of the bounds. What follows is a proof of their results consistent with the notation developed here.

Theorem 5.3 *Let $G = G_1 + \ldots + G_n$. For all t, Left has a strategy such that he can guarantee:*

$$\begin{aligned} \alpha: &\ {}^tV_R(G) \geq {}^tV_R(G_1) + \cdots + {}^tV_R(G_n) - t \\ \beta: &\ {}^tV_L(G) \geq {}^tV_L(G_1) + {}^tV_R(G_2) + \cdots + {}^tV_R(G_n) \end{aligned}$$

Proof: First, assume that part β is true. To show that part α follows, assume that Right moves in G_1. By induction:

$$\begin{aligned} {}^tV_R(G) &= {}^tV_L({}^tR_1^{0-}(G_1) + G_2 + \cdots + G_n) \\ &\geq {}^tV_L({}^tR_1^{0-}(G_1)) + {}^tV_R(G_2) + \cdots + {}^tV_R(G_n) \end{aligned} \tag{5.4}$$

By the dual of equation 2.3, and since the value of a game can only increase if Right plays non-optimal moves:

$${}^tV_R(G_1) - t = {}^tV_L({}^tR_1(G_1)) \leq {}^tV_L({}^tR_1^{0-}(G_1))$$

Replacing ${}^tV_L({}^tR_1^{0-}(G_1))$ with ${}^tV_R(G_1) - t$ in Equation 5.4 yields:

$${}^tV_R(G) \geq ({}^tV_R(G_1) - t) + {}^tV_R(G_2) + \cdots + {}^tV_R(G_n)$$

Now assume that part α is true. Left has a strategy that guarantees β. In particular, Left makes an optimal move in some game, lets say G_1, such that $\sigma(G_1) \leq t$. By induction:

$$\begin{aligned} {}^tV_L(G) &= {}^tV_R({}^tL_1^{0-}(G_1) + G_2 + \cdots + G_n) \\ &\geq {}^tV_R({}^tL_1^{0-}(G_1)) + {}^tV_R(G_2) + \cdots + {}^tV_R(G_n) - t \end{aligned} \tag{5.5}$$

Since $\sigma(G_1) \leq t$, by equation 2.3:

$${}^tV_L(G_1) = {}^tV_R({}^tL_1^{0-}(G_1)) - t$$

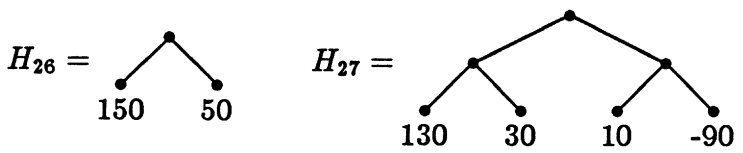
So replacing ${}^tV_L(G_1)$ for ${}^tV_R({}^tL_1^{0-}(G_1)) - t$ in Equation 5.5 yields:

$${}^tV_L(G) \geq {}^tV_L(G_1) + {}^tV_R(G_n) + \cdots + {}^tV_R(G_n)$$

If there are no component game G_1 for which $\sigma(G_1) > t$ then the above argument does not hold. However, in that case β by Equation 2.6 reduces to:

$$mean(G_1 + \cdots + G_n) = mean(G_1) + \cdots + mean(G_n)$$

The linearity of the mean value function was proved by Hanner[7]. ∎



$Mean(H_{26}) = 45$ $\quad$ $Mean(H_{27}) = 20$

$\sigma(H_{26}) = 55$ $\quad$ $\sigma(H_{27}) = 60$

${}^0V_L H_{26} = 50$ $\quad$ ${}^0V_L H_{27} = 30$

${}^0V_R H_{26} = 45$ $\quad$ ${}^0V_R H_{27} = 10$

Figure 5.8: Defining H_{26} and H_{27}

5.1.5 Comparing Results

The thermostatic strategy and the *second highest* σ result both improve upon Hanner's result. Often they produce the same bounds. For example, they both produce the same bounds for $H_{23} + H_{24} + H_{25}$ where H_{23}, H_{24}, and H_{25} are defined in Figure 5.7.

However, there exist sums of games for which each result is superior. For example, consider the games shown in Figure 5.8. The *second largest* σ method can only bound the value of the game to within 55. The thermostatic strategy produces the best bounds when $t = 0$, in which case:

$$\begin{array}{lcll} {}^0V_L H_{26} + {}^0V_R H_{27} & \leq & V_L(H_{26} + H_{27}) & \leq & {}^0V_L H_{26} + {}^0V_L H_{27} + t \\ 50 + 10 & \leq & V_L(H_{26} + H_{27}) & \leq & 50 + 30 + 0 \\ 60 & \leq & V_L(H_{26} + H_{27}) & \leq & 80 \end{array}$$

The accuracy of the bounds is 20.

On the other hand, cases exist when the *second highest* σ produces better bounds. For example, consider the games shown in Figure 5.9 [2].

[2] A slightly more complex example that has exactly the same behavior is

$$\{\, 25, \{\, 50 \mid 0 \,\} \mid -75 \,\} + \{\, 20 \mid -20 \,\}$$

$Mean(H_{28}) = -25$ $\quad$ $Mean(H_{29}) = 0$
$\sigma(H_{28}) = 50$ $\quad$ $\sigma(H_{29}) = 20$
${}^0V_L H_{28} = 25$ $\quad$ ${}^0V_L H_{29} = 20$
${}^0V_R H_{28} = -25$ $\quad$ ${}^0V_R H_{29} = -20$

Figure 5.9: Defining H_{28} and H_{29}

The *thermostatic* strategy states that the best bounds are obtained when $t = 0$. In particular:

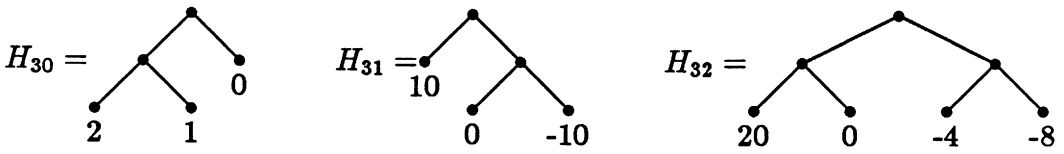
$$\begin{array}{lcll} {}^0V_L H_{28} + {}^0V_R H_{29} & \leq & V_L(H_{28} + H_{29}) & \leq & {}^0V_L H_{28} + {}^0V_L H_{29} + t \\ 25 + -20 & \leq & V_L(H_{28} + H_{29}) & \leq & 25 + 20 + 0 \\ 5 & \leq & V_L(H_{28} + H_{29}) & \leq & 45 \end{array}$$

So, the accuracy of the bounds is 40. However, using the *second largest* σ result, it is possible to bound the value of the game to within 20.

5.2 Search

Searching for the optimal solution to an instance of SUM requires exponential time. However, good pruning techniques can greatly cut down the time required to perform a search. In particular, the approximate solutions presented in this chapter provide a great deal of power in terms of pruning and directing the search for an optimal solution.

One simple example occurs in an alpha-beta pruning search. The effectiveness of the alpha-beta pruning techniques is highly dependent on the order that the nodes are searched [12]. The approximate algorithms can be used to place an ordering on the nodes. One useful heuristic, for example, is to first explore the search tree below the node that has the highest possible pay off.



$Mean(H_{30}) = 1$ $\sigma(H_{30}) = 1$ $Mean(H_{31}) = 7.5$ $\sigma(H_{31}) = 7.5$ $Mean(H_{32}) = 0$ $\sigma(H_{32}) = 6$

Figure 5.10: Defining H_{30}, H_{31}, and H_{32}

However, the alpha-beta pruning technique does not fully use the information provided by the approximate solutions. For example, assume its Left's move in the sum of games shown in Figure 5.10. Applying Theorem 5.1 [3] to each of Left's options, yields:

$$\begin{array}{rcccl} 9 & \leq & V_L(L_1 H_{30} + H_{31} + H_{32}) & \leq & 16.5 \\ 11 & \leq & V_L(H_{30} + L_1 H_{31} + H_{32}) & \leq & 17 \\ 18.5 & \leq & V_L(H_{30} + H_{31} + L_1 H_{32}) & \leq & 28.5. \end{array}$$

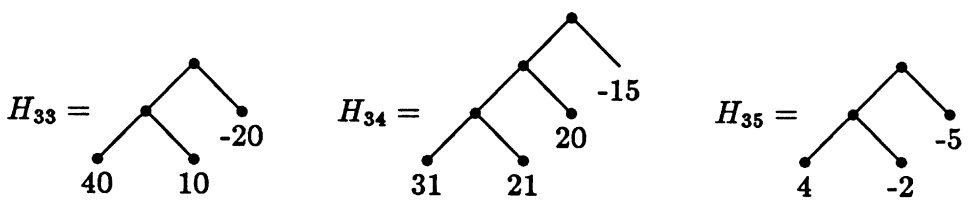
It becomes immediately apparent that playing in H_{32} is optimal. However, an alpha-beta search would expand out the search tree in order to prove that H_{32} is optimal.

Theorem 5.1, does not guarantee that it will uniquely distinguish one move as superior to all others. However, it does give valuable information about the relative merit of each move. One effective use of the information supplied by the approximate algorithms occurs in the B^* algorithm [4]. What follows is a description of the algorithm.

5.2.1 The B^* Algorithm

Each node in the search tree generated by B^* search is associated a range of values such that the actual value of the node is guaranteed to be within that

[3] Though Theorem 5.2 and and Theorem 5.3 produce better bounds, the bounds produced by Theorem 5.1 are easier for the author to produce and for the reader to verify. Theorem 5.1 will be used exclusively through out this section.



$Mean(H_{33}) = 2.5$
$\sigma(H_{33}) = 22.5$

$Mean(H_{34}) = 3$
$\sigma(H_{34}) = 18$

$Mean(H_{35}) = -2$
$\sigma(H_{35}) = 3$

Figure 5.11: Defining H_{33}, H_{34}, and H_{35}

range. This information is used both to direct the search, and to terminate the search promptly.

At any point in the search, the range of values associated with a node has been determined via an evaluation function like Theorem 5.1, or have been derived by backing up information obtained through exploration of the search tree below the given node. For example, assume that the search begins with Left to move in the sum of games defined in Figure 5.11. By Theorem 5.1, $3 \leq V_L(H_{33} + H_{34} + H_{35}) \leq 26$. However, after exploring the search tree as shown in Figure 5.12, it becomes apparent that, at worst, Left can move to $\{ 40 \mid 10 \} + H_{34} + H_{35}$ and get 8. Hence, the bounds can be improved to be: $8 \leq V_L(H_{33} + H_{34} + H_{35}) \leq 26$.

The basic B^* algorithm consists of expanding a leaf of the search tree, using the evaluation function to compute the range of that node, and then backing up the new information. Assuming the search is looking for the optimal move for Left, the search is terminated when the lower bounds on one of Left's options is greater than or equal to the upper bound of all the other options.

The key question in B^* search is which node on the frontier to expand. What distinguishes B^* from other search strategies, like best-first, is that B^* employs two different strategies. The first strategy, called *Disprove-Rest*, deepens the search tree under the second best node trying to decrease its upper bound. The other strategy, called *Prove-Best*, deepens the search tree under the "best" candidate node trying to increase that nodes lower bound.

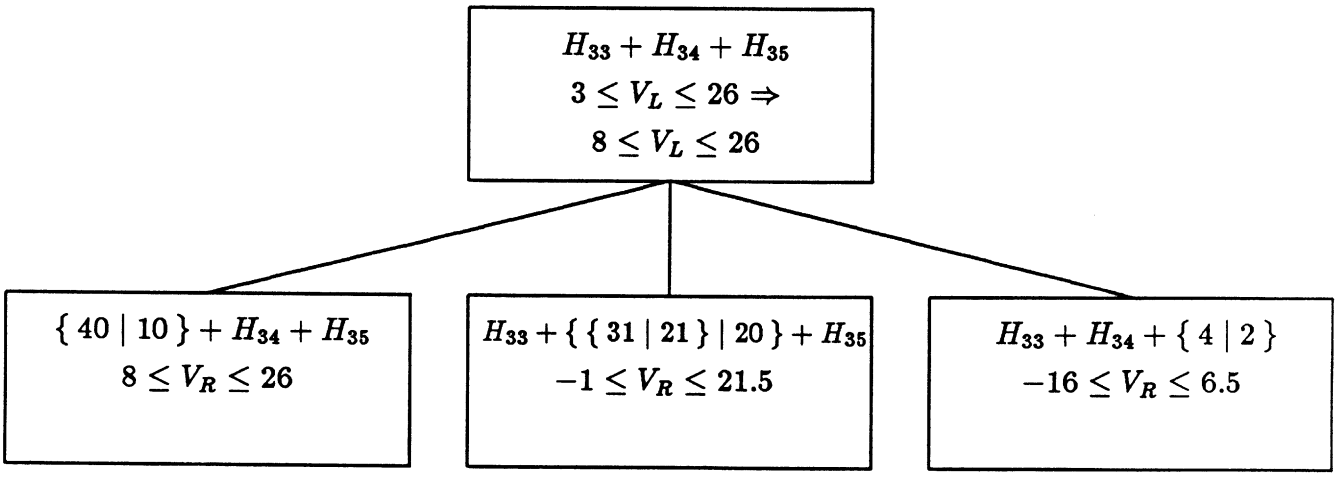


Figure 5.12: Expanding the Search Tree One Level

The power of the B^* search comes from its ability to use both strategies. In game situations, it is often easier to prove that a move is bad, then to prove that a move is good. For example, Figure 5.13 shows the search tree in Figure 5.12 expanded one level using the *Disprove-Rest* strategy. The expansion lowers the runner-up's upper bound from 21.5 to 2. This is enough to terminate the search, since the best nodes lower bound (8) is now greater than the other nodes upper bounds (2, 6.5). However, if the tree in Figure 5.12 is explored using *Prove-Best*, then the search does not terminated as quickly. Figure 5.14 shows the best-node expanded one level. The expansion produces tighter bounds for the best node, but does not produce a lower bound greater than 21.5.

Berliner [4] suggested two rules for choosing between *Prove-Best* and *Disprove-Rest*. The first favored exploration of subtree which had not been explored deeply. The second favored the exploration of nodes with large ranges. Palay [19] obtained better results by assuming that the actual value of a node is uniformly distributed over the range of the node, and then the strategy with the highest probability of success was chosen. Palay [20] improved upon his previous work to allow for a arbitrary probability distribution, as oppose to assuming uniform distributions.

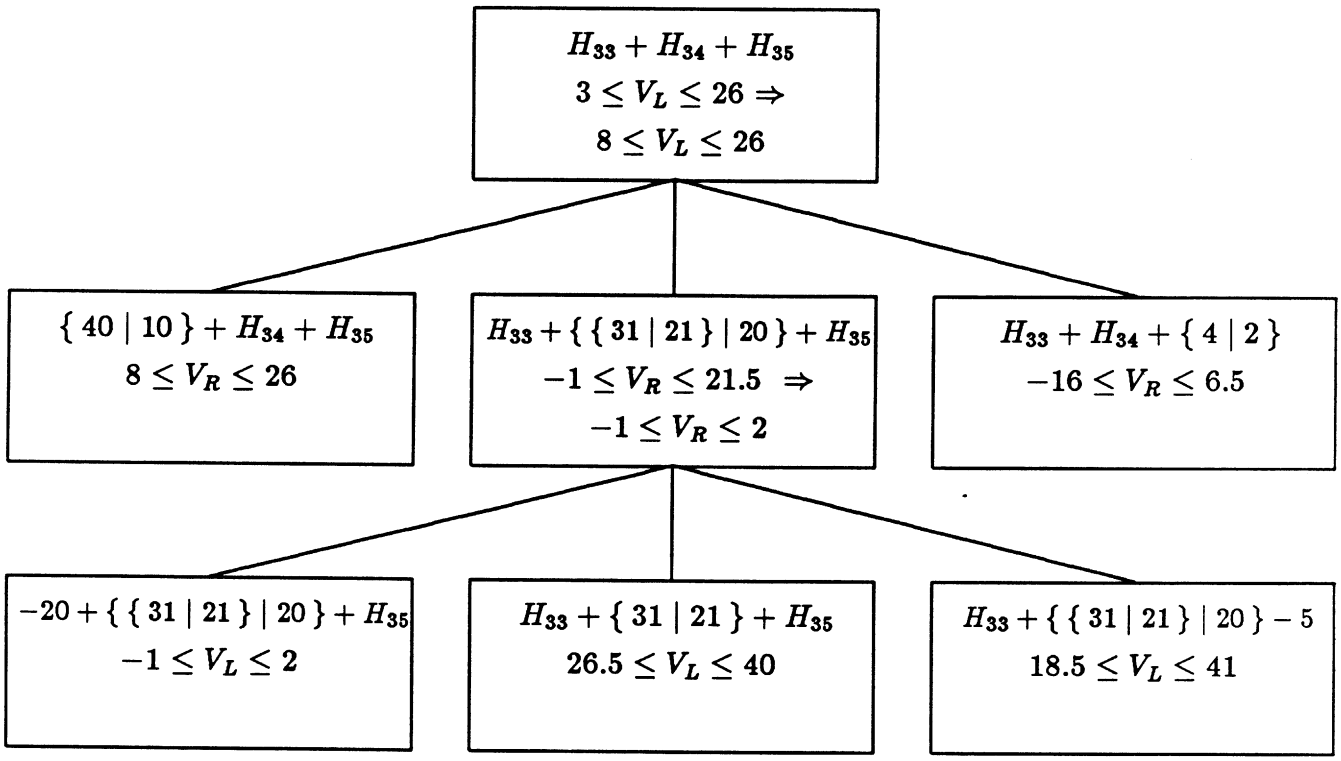


Figure 5.13: The Search Tree Expanded using *Disprove-Rest*

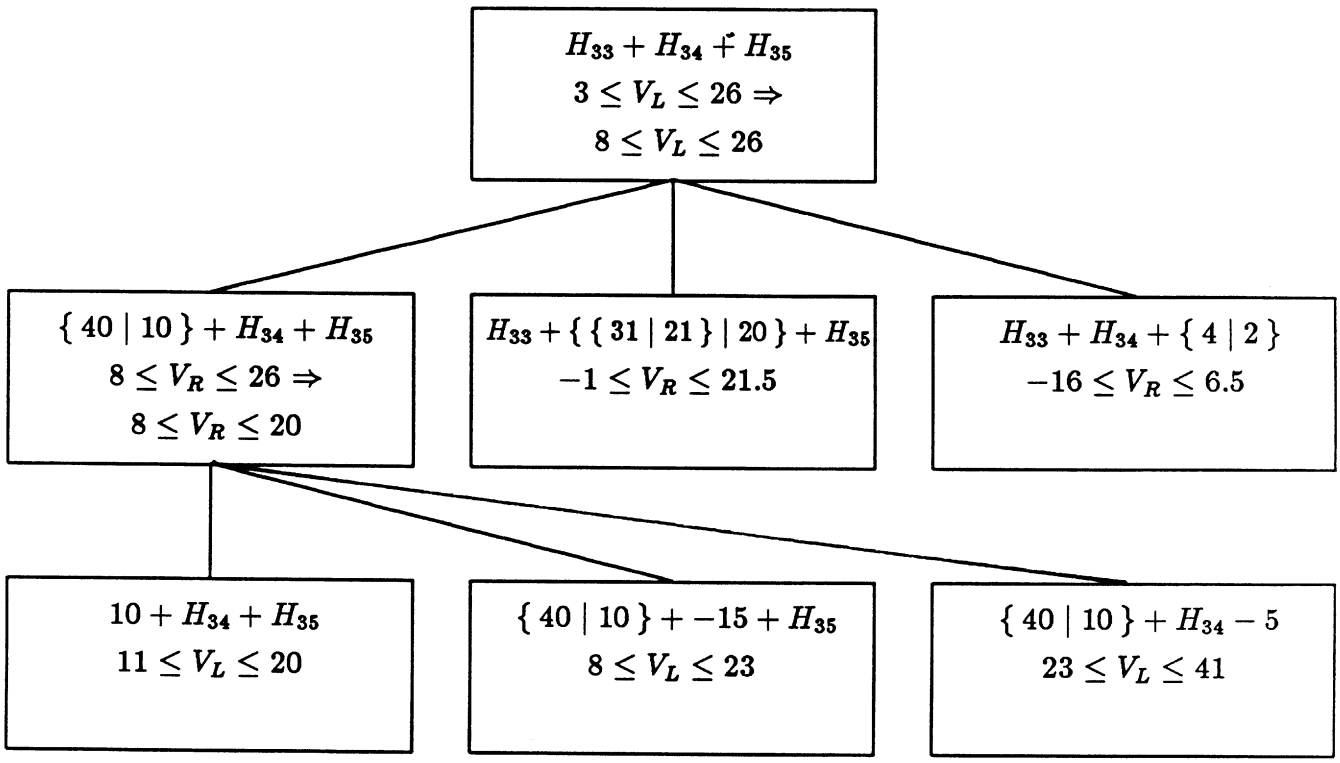


Figure 5.14: The Search Tree Expanded using *Prove Best*

Chapter 6

Endgame of Go

The endgame of Go is naturally characterized as a sum of games. During a game of Go, the players place black and white stones down on a square grid trying to surround territory for themselves and to invade the potential territory of their opponent. In the beginning and middle game, it is very common for a move to have both a local and global effect. However, towards the end of the game, the board becomes divided up into a number of separate regions and each move has only a local effect.

For example, Figure 6.1 shows a simplistic Go position that has reached the final stages of the endgame. Black has taken a large center territory while white has carved out four separate corner territories. The rest of the game revolves around a number of small, distinct, border disputes.

The endgame of Go is an important example of a sum of games because of the wide spread interest in the game. Go is played professionally in the Orient at the level that chess is played in the West. It is common for a game between professionals to be very close (within one point) and hence the endgame has been studied extensively, *e.g.*, [18].

Furthermore, Go has attracted a reasonable amount of attention from the A.I. community. Unlike Chess, Go does not simply succumb to brute force search. For that reason, Berliner [3] said that "even if a full-width search program were to become World Chess Champion, such an approach cannot possibly work for Go, and this game may have to replace chess as the task par excellence for A.I." Interesting work has been done on constructing Go playing programs, *e.g.*, [1], [13], [23], and [24].

Though the endgame of Go is naturally described as a sum of games,

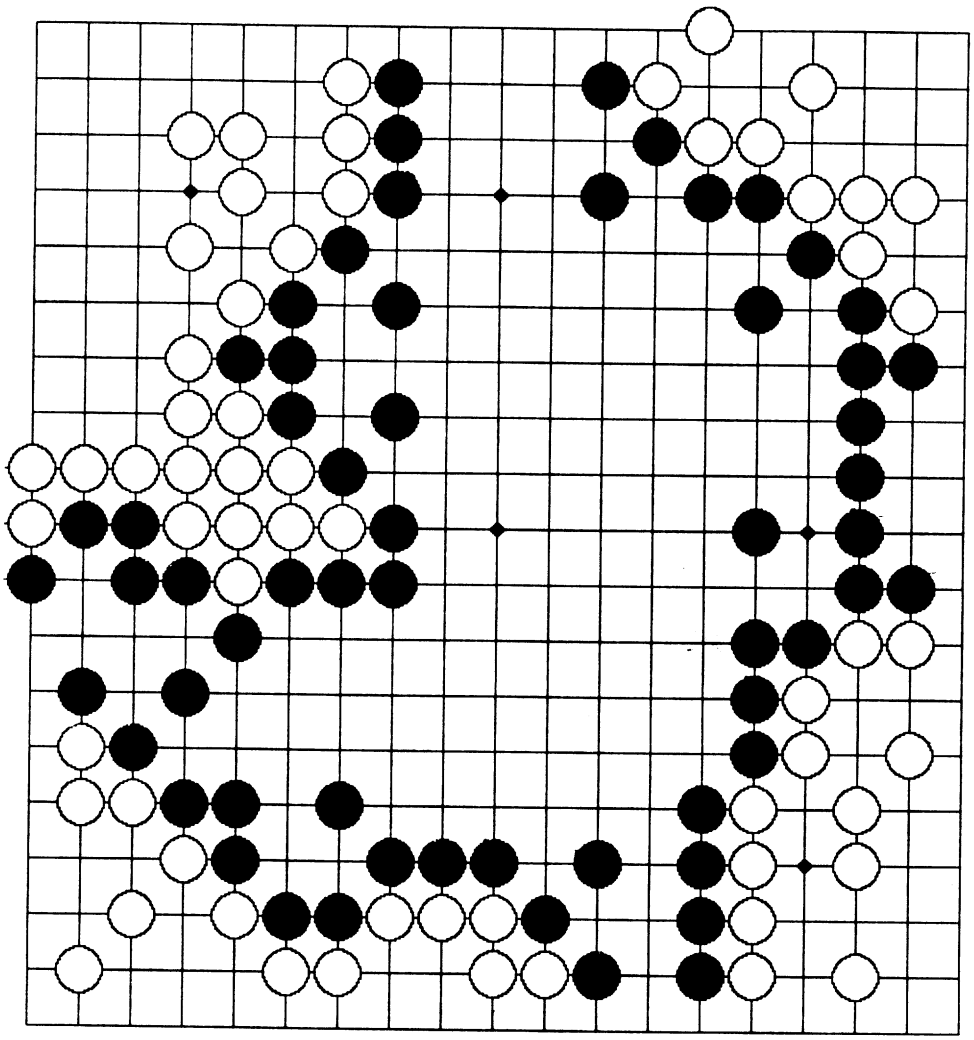
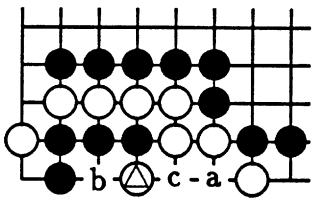


Figure 6.1: An end position in a game of Go



Assuming its Black's turn to move, optimal play is the following: Black plays at a, White responds at b, Black plays at c taking two white stones, and White responds at △ capturing two black stones. At which point, the position has repeated itself. Under Japanese rules the game is considered a draw.

Figure 6.2: *Chōsei:*

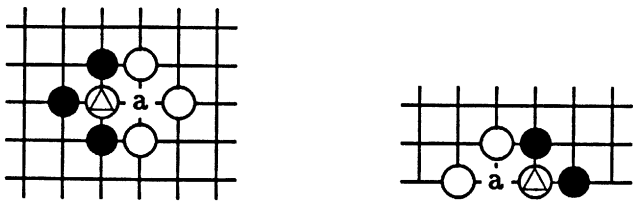
issues arise when the theory of a sum of games is applied to the endgame of Go. This chapter address those issues.

This chapter assumes that the reader is familiar with the basic rules of Go. A good introduction to the game can be found in *How to Play Go* by Takagawa[28].

6.1 Ko

Repetitive situations requiring special rules often occur in board games. For example, in Chess, if a board position is repeated three times, then the rules state that the game is a draw. Similarly, using the Japanese Go rules, some repetitive situations result in a drawn game. Such situations are rare, but do exist. For example, Figure 6.2 shows one such situation called *chōsei* meaning eternal life[29].

One instance of a potentially repetitive situation that often occurs in Go is called *ko*. The two typical configurations of a ko are shown in Figure 6.3. Without a special rule, both positions could result in an infinite capture - recapture sequence. The ko rule prevents an infinite repetitive sequence by stating that if one player takes a ko, then his opponent can not retake the ko on the next turn.



If Black plays at a, *then by the ko rule, White can not play at* ⃝ *on the next turn*

Figure 6.3: Ko

Consider a ko position as in Figure 6.3. When it is important to the players whether Black ends up with a stone at *a* or whether White ends up with a stone at ⃝ a *ko fight* ensues. The basic form of a ko fight is the following: Black takes the ko. Unable to immediately retake due to the ko rule, White plays a *ko threat*, *i.e.*, a threatening move elsewhere on the board. Black, unwilling to watch White carry through on the threat, responds to the ko threat. White then takes the ko. Black is now in the position White was at the start of the fight. Black wants to take the ko but is unable to because of the ko rule. So, Black makes a ko threat. The ko fight continues until one player ignores his opponent's ko threat and fills the ko.

Ko fights are an important part of the game of Go. They occur frequently in the endgame. Unfortunately, the nature of kos and of a ko fight is very different than the games considered thus far.

The ko rule, along with other special case rules, prevent repetitive board positions. However, when viewed in a local situation, a ko is a repetitive situation. In particular, assume that the endgame of Go is represented as a sum of small border disputes and assume that one of the border disputes involves a ko. During the ko fight, the border position containing the ko will repeatedly oscillate between two positions, *i.e.*, Black has the ko, or White has the ko.

This cyclic nature, prevents ko from being represented as a finite Conway game. One could imagine representing ko as an infinite tree or as a graph. However, such a representation is outside of the theory as currently

known.

The notion of mean value and temperature of a game is crucial to the heuristic solutions developed in section 5.1. However, these notions do not make sense in relationship to ko. The computation of mean value and temperature, via thermographs, is dependent upon the representation of a game as a finite tree.

6.2 Complexity Results

Lichtenstein and Sipser[15] proved that Go is $PSPACE$-hard. At first glance it would appear that one could prove that the endgame of Go is also $PSPACE$-hard by using theorem 4.2 and constructing a reduction from $SUM_{d\leq 2}$ to the endgame of GO [1]. However, two difficulties arise with this approach.

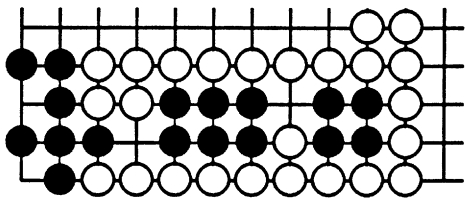
The first difficulty is in constructing a reduction from $SUM_{d\leq 2}$ to the endgame of GO. It is easy to construct a Go position that has the structure of a switch or a left heavy tree, *e.g.*, Figure 6.4. However, to transform more complex games into Go positions is hard. It is not at all clear that the games used in the reduction from QBF to $SUM_{d\leq 2}$ correspond to Go positions.

The basic problem is that a Go position has a great deal of structure that is not part of an arbitrary game. For example, in a Go position almost all the places Black can play White can play. It is hard to imagine a Go position where Black has 100 valid moves but White only has one. Hence, in any game corresponding to a Go position, Left and Right will have approximately equal number of options.

Similarly, the persistence of moves in a Go position greatly restricts the types of games that can be represented on a Go board. Generally speaking, if a move is initially available for Black, then the move remains available for Black until one players places a stone at that point. This type of structure is uncommon.

The games used in the reduction from QBF to $SUM_{d\leq 2}$ in Theorem 4.2 are switches, left heavy trees, and the game shown in Figure 6.5. Switches

[1] The comments made in this section also apply to Morris's [17] proof that SUM is $PSPACE$-complete



Assume that Left is playing white, and that the white group is alive. Then, this position represents the tree:

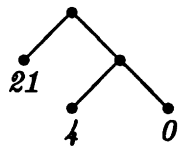


Figure 6.4: Representing a Left Heavy Tree

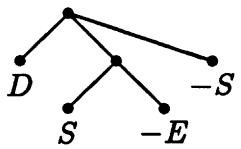


Figure 6.5: A Key Game in the $PSPACE$-complete Proof

and left heavy trees are easy to represent on a go board. However, there is strong reason to believe that the game shown in Figure 6.5 can not be represented on a Go board.

The game in Figure 6.5 seems to violates Go's persistence of moves. Right has two moves. Right can make a very large threat or Right can take a quick profit. What's unusual, however, is that if Right takes a quick profit, then he can no longer make the threat. That is, the move that takes the quick profit must also stabilize, invalidate, or in some way remove the other move. Similarly, if Right takes the quick profit, he can no longer make the threat. Constructing such a Go position seems unlikely.

Even if one had a reduction from $SUM_{d\leq 2}$ to the endgame of GO, a problem remains. If the standard representation of a GO position is used, then the chain of reductions from QBF to $SUM_{d\leq 2}$ as in Theorem 4.2 to the endgame of GO does not correspond to polynomial time reduction. The heart of the problem is that Go positions are unary representations, and the reduction from QBF to $SUM_{d\leq 2}$ uses position numeric notation.

In particular, the standard representation of a Go position as a n x n square grid implies that the size of a Go position is proportional to its value. For example, on a square grid, a position worth 30 corresponds to position corresponds to either 30 open grid points, 15 grid points containing 15 prisoners, or some other combination of open points and prisoners. Furthermore, though the reductions in Theorem 4.2 create an instance of $SUM_{d\leq 2}$ that is only polynomially larger than the initial input, the value of the numbers in the constructed game are exponentially larger than the size of the original input. Hence, if the reduction from QBF to the endgame of Go used Theorem 4.2, then the resulting Go position would be exponential larger than size of the original input.

Morris[17] conjectures that this problem can be solved through the use of a compact notation for fractional values. He comments that Go positions "can exhibit almost any form of composition from dyadic rationals." However, though fractional values are used to *estimate* Go positions, fractional values *never* actually appear in Go. The leaf values of any game tree corresponding to a Go position are always integral. Hence, fractional notation will not solve the problem.

The problem might be solved if a compact representation of Go positions was allowed. For example, one could imagine a representation where

the territory surrounded by a group was represented by a binary number, instead of by the corresponding number of empty grid points.

6.3 Approximation and Search Algorithms

Ignoring ko, the approximate solutions and search algorithms discussed in this thesis are applicable to the endgame of Go. One could imagine a Go playing program that represented each of the border disputes as a tree, and then determined its next move by Hanner's heuristics, or by B^* search combined with Hanner's approximations.

The biggest problem facing such a Go playing program, is simply extracting the game trees from the Go board. The success of the extraction depends upon the programs ability to recognize the decomposition of the board into a number of separate border disputes, determine the life and death of groups, and determine the connectivity of loosely related stones. All these tasks are non-trivial.

Chapter 7

Future Research

This thesis analyzes play in a sum of games from three different perspectives: computational complexity, heuristic solutions, and optimal search algorithms. This chapter summarizes the results and considers directions for future research.

7.1 Complexity

Determining the optimal strategy in a sum of games seems to be hard. Lockwood Morris[17] proved that SUM is $PSPACE$-complete. This thesis proves that even when the component games in the sum are very small, *i.e.*, maximum depth of two, the problem remains $PSPACE$-complete. However, there may be interesting classes of sums of games that, even assuming $P \neq PSPACE$, can be played optimally in polynomial time. Future research is needed to determine exactly when the problem is $PSPACE$-complete.

For example, both Morris's $PSPACE$-completeness proof and the proof of Theorem 4.2 rely on a game that has *multiple options*, *i.e.*, the proofs rely on a game in which Right is given a choice between two moves. The question arises: does the difficulty of playing a sum of games depend upon having multiple options. My conjecture, after spending a large amount of time unsuccessfully trying to determine optimal play in a sum of left heavy trees, is that SUM is $PSPACE$-complete even when the component games have maximum depth two and have no multiple options.

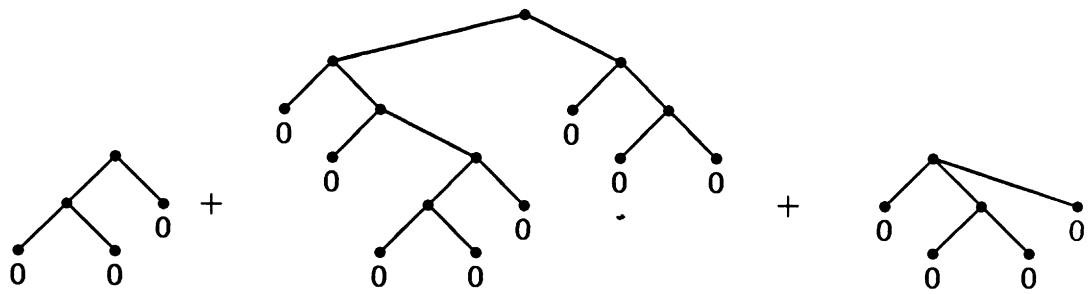


Figure 7.1: A Sum of All Small Games

An interesting class of games are the *all small*[2] games. A game is all small if all the leaf values are zero, and both players have legal moves from every non-terminal position. Thus, no subtree of the game represents a number. For example, Figure 7.1 shows a sum of all small games.

Determining the optimal strategy in a sum of all small games seems hard. Some knowledge about how to play all small games can be obtained through Berlekamp, Conway, and Guy's theory of *atomic weights*[2]. However, I conjecture that the problem of determining the winning strategy in sum of all small games is also $PSPACE$-complete.

The possibility exists that SUM could be solved using a pseudo-polynomial time algorithm. The question is whether there exists a natural set of restrictions that can be applied to instances of SUM to yield a polynomial algorithm. For example, does limiting the size of the values of the terminal positions allow for a polynomial time algorithm? Unfortunately, given the difficulty of playing all small games, there is a real possibility that SUM is hard in a strong sense. Search for a pseudo-polynomial time algorithm and a NP-hardness proof in a strong sense should proceed in parallel.

7.2 Heuristic Solutions

Hanner[7] proved that the final score of a sum of games can be approximated to within the maximum temperature of a component game. This result is quite remarkable. It bounds the value of a sum of games in a way that is independent of the number of games in the sum and independent of the complexity of the component. Though this thesis presents a clear proof, a simple, short, intuitive argument for why the final score can be approximated to within the largest temperature is still needed.

Central to Hanner's proof is the concept of taxation. An important property of taxation is that it is linear. Let $G = G_1 + \ldots + G_n$ and let each component game G_i be approximated by the taxed version of that game. Then one way to view Hanner's proof is that G is approximated as the sum of the approximated component games.

In the search for better heuristics, a natural way to proceed is to consider other linear approximation functions besides taxation. However, it is surprisingly hard to find linear functions. The mean function and any multiple of the mean function is linear. Also, in [2], a linear function, called overheating, is defined. However, the simpler heating function is not linear.

All the functions mentioned above, *i.e.*, taxation, mean, heating, overheating, have the form

$$F(\{L \mid R\}) = \begin{cases} c * (\{L - t \mid R + t\}) & \text{if P(G,...)} \\ N(G,\ldots) & \text{otherwise} \end{cases}$$

for some value of c, some predicate P, and some function N. It would be useful to known the constraints on P and N such that the above form produces a linear function.

Perhaps even more amazing than Hanner's bounds is Hanner's mean strategy. Hanner proved if a player uses the mean strategy in

$$G = G_1 + \ldots + G_n,$$

then he can force the final score to be within the $Max(\sigma(G_i))$ of the optimal score. This thesis proved that, by using the mean strategy, a player can force the final score to be within the second largest $\sigma(G_i)$. However, by using the mean strategy, a player often does even better than that. How good is the mean strategy?

7.3 Search

Searching for the optimal solution to an instance of SUM requires exponential time. However, the approximate solutions presented in this thesis provide a great deal of power in terms of pruning and directing the search for an optimal solution. In particular, the thesis suggests that the approximate solutions combined with Berliner's B^* search algorithm will reduce the time required to find the optimal solution.

One direction for future research is to quantitatively measure how efficient B^* is. On average, what improvement does one gain using a B^* instead of alpha-beta search algorithm? Is there a simple way to characterize those games that are solved efficiently using B^*?

The power of the B^* search comes from its use of both a *prove best* and a *disprove rest* strategy. In the context of solving an instance of SUM, are there any good heuristics for choosing one strategy over another?

Bibliography

[1] Benson, D., Hilditch, B. R., and Denbigh, S. [1979] "Tree Analysis Techniques in Tsumego" *Proceedings of 6th International Joint Conference on Artificial Intelligence*, Tokyo.

[2] Berlekamp, E. R., J. H. Conway, and R. K. Guy [1982], *Winning Ways*, Academic Press Inc., London.

[3] Berliner, H. [1978], "A Chronology of Computer Chess and its Literature", *Artificial Intelligence*, Vol 10, Number 2.

[4] Berliner, H. [1979], "The B* Tree Search Algorithm: A Best-First Proof Procedure", *Artificial Intelligence*, Vol 12, pp. 23-40.

[5] Conway, J. H. [1976], *On Numbers and Games*, Academic Press Inc., London, New York and San Francisco.

[6] Conway, J. H. [1985] Personal Communications.

[7] Hanner, O. [1959] "Mean play of sums of positional games", *Pacific J. Math.* Vol 9, pp. 81-99.

[8] Fraenkel, A. S. and Lichtenstein, D. [1981] "Computing a Perfect Strategy for nxn Chess Requires Time Exponential in n" *Journal of Combinatorial Theory* Series A 31, pp. 199-214.

[9] Johnson, D. S., [1983]"The NP-Completeness Column: An Ongoing Guide", *Journal of Algorithms* Vol 4, pp. 397-411.

[10] Karp, R.M. [1972], "Reducibility among combinatorial problems," *Complexity of Computer Computations*, R. E. Miller and J. W. Thatcher (eds), Plenum Press, N.Y., pp. 85-103.

[11] Knuth, D. E. [1974], *Surreal Numbers*, Addison-Wesley Publishing Company, Inc.

[12] Knuth, D. E. and Moore, R. W. [1975] "An Analysis of Alpha-Beta Pruning", *Artificial Intelligence*, Vol 6, Number 4, pp. 293-326.

[13] Lehner, P. E., *Planning in Adversity: A computational Model of Strategic Planning in the Game of Go* [1981] Ph.D. Thesis, University of Michigan.

[14] Nilsson, N. J. [1980], *Principles of Artificial Intelligence*, Tioga Publishing Company, California.

[15] Lichtenstein, D. and Sipser, M. [1978] "Go is Pspace Hard", *Proceedings of the Nineteenth Annual Symposium on Foundations of Computer Science*, Ann Arbor, Michigan.

[16] Milnor, J. [1953], "Sums of Positional Games", *Contributions to the Theory of Games, Ann. Math Studies # 28* editors Kuhn and Tucker, pp. 291-301.

[17] Morris, F. L., "Playing Disjunctive Sums is Polynomial Space Complete", *Int. Journal of Game Theory* Vol. 10, Issue 3/4, pp. 195-205.

[18] Ogawa, T. and J. Davies [1976], *The Endgame*, Ishi Press Inc., Tokyo.

[19] Palay, A. J. [1982], "The B^* Tree Search Algorithm - New Results", *Artificial Intelligence*, Vol 19, Number 2, pp.145-163.

[20] Palay, A. J. [1983] *Searching with Probabilities* Ph. D. Thesis, Carnegie-Mellon University, Pittsburgh, Pa.

[21] Propp, J. [1983] "Nim for Three - An Overview and an Offer of Alcohol" *Eureka* Cambridge University, England.

[22] Reif, J. H. [1984] "The Complexity of Two-Player Games of Incomplete Information" *Journal of Computer and System Sciences* Vol 29. pp. 274-301.

[23] Reitman, W., Nado, R., and Wilcox, B. [1978] "Machine Perception: What Makes It So Hard for Computers to See?" *Perception and Cognition: Issues in the Foundations of Psychology* Minnesota Studies in the Philosophy of Science, Vol 9., University of Minneapolis Press.

[24] Reitman,W., and Wilcox, B. [1979] "The structure and performance of the Interim.2 Go Program", *Proceedings of 6th International Joint Conference on Artificial Intelligence*, Tokyo.

[25] Schafer, T. J. [1978] "On the Complexity of Some Two-Person Perfect-Information Games" *Journal of Computer and System Sciences* Vol 16, pp. 185-225.

[26] Stockmeyer, L. J. and Chandra, A. K. [1979] "Provably Difficult Combinatorial Games" *SIAM J. Comput.* Vol 8, No 2, pp. 151-174.

[27] Stockmeyer, L. J. and Meyer, A. R. [1973] "Word Problems Requiring exponential Time: Preliminary Report", *Proceedings of the 5th Anual ACM Symposium on Theory of Computation*, Association for Computing Machinery, New York.

[28] Takagawa, S. [1956] *How to Play Go*, The Nihon Ki-in, Japan.

[29] Tilley, J.S. [1970] *Go: International Handbook and Dictionary*, Ishi Press, Tokyo.

This blank page was inserted to preserve pagination.

CS-TR Scanning Project
Document Control Form

Date : 3/30/95

Report # LCS-TR-348

Each of the following should be identified by a checkmark:
Originating Department:

☐ Artificial Intellegence Laboratory (AI)
☒ Laboratory for Computer Science (LCS)

Document Type:

☒ Technical Report (TR) ☐ Technical Memo (TM)
☐ Other:_____

93

Document Information

Number of pages: 98 (100 IMAGES).
Not to include DOD forms, printer instructions, etc... original pages only.

Originals are:
☒ Single-sided or
☐ Double-sided

Intended to be printed as :
☐ Single-sided or
☒ Double-sided

Print type:
☐ Typewriter ☐ Offset Press ☐ Laser Print
☐ InkJet Printer ☒ Unknown ☐ Other:_____

Check each if included with document:

☐ DOD Form ☐ Funding Agent Form ☒ Cover Page
☒ Spine ☒ Printers Notes ☐ Photo negatives
☐ Other: _____

Page Data:

Blank Pages(by page number):_____

Photographs/Tonal Material (by page number):_____

Other (note description/page number):

Description :	Page Number:
IMAGE MAP (1) UNNUMBERED TITLE PAGE	
(2-7) PAGES #'ED ii-vii	
(8-93) PAGES #'ED 1-86	
(94-97) SCANCONTROL, COVER, SPINE, PRINTERS NOTES	
(98-100) TRGTS(3)	

Scanning Agent Signoff:
Date Received: 3/30/95 Date Scanned: 4/10/95 Date Returned: 4/11/95

Scanning Agent Signature: Michael W. Cook

Scanning Agent Identification Target

Scanning of this document was supported in part by the **Corporation for National Research Initiatives,** using funds from the **Advanced Research Projects Agency** of the **United states Government** under Grant: **MDA972-92-J1029.**

The scanning agent for this project was the **Document Services** department of the **M.I.T Libraries.** Technical support for this project was also provided by the **M.I.T. Laboratory for Computer Sciences.**

Scanned

Date: 4/10/1995

M.I.T. Libraries
Document Services