

Applying Integer Programming Techniques to Find Minimum Integer Weights of Voting Games

by

Aaron B Strauss

Submitted to the Department of Electrical Engineering and Computer
Science

in partial fulfillment of the requirements for the degree of

Master of Engineering in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2003

© Massachusetts Institute of Technology, 2003. All rights reserved.

Author

Department of Electrical Engineering and Computer Science

July 18, 2003

Certified by

Stephen Ansolabehere

Professor, Department of Political Science

Thesis Supervisor

Accepted by

Arthur C. Smith

Chairman, Department Committee on Graduate Students

Applying Integer Programming Techniques to Find Minimum Integer Weights of Voting Games

by

Aaron B Strauss

Submitted to the Department of Electrical Engineering and Computer Science
on July 18, 2003, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Computer Science and Engineering

Abstract

Using concepts from computer science and mathematics I develop three algorithms to find the minimum integer weights for voting games. Games with up to at least 17 players can be solved in a reasonable amount of time. First, coalitions are mapped to constraints, reducing the problem to constraint optimization. The optimization techniques used are Gomory's all integer simplex algorithm and a variant of the popular integer programming method branch and bound. Theoretical results include that minimum integer weights are not unique and a confirmation of a prior result that minimum integer weights are proportional to *a priori* seat share. Thus, these algorithms can be useful for researchers evaluating the differences between proportional bargaining models and *formateur* models. The running times of the different algorithms are contrasted and analyzed for potential improvements.

Thesis Supervisor: Stephen Ansolabehere
Title: Professor, Department of Political Science

Acknowledgments

Before I thank the individuals whose guidance has been invaluable, I especially need to thank the Institute for providing me with the tools and opportunity to learn, teach, laugh, and live over these incredible five years. The creative spirit and genius of the student body, combined with constant skirmishes with the administration produced stories to tell for decades to come. To everyone I met along the journey, thank you for the indelible memories; i will miss you.

In my previous thesis, I thanked Professor Ansolabehere; it is now more accurate to say that I am indebted to him. His constant communication and invaluable guidance throughout the entire process saved me countless hours of fruitless labor. Professor Snyder had numerous insights on the specifics of the algorithm. Jonathan Kennell introduced me to integer programming, and my brother, Marc, augmented my understanding of the simplex method. Everyone involved in the process, from Tobie Weiner to Professor Peterson, has helped me learn so much about a topic I knew nothing about just one year ago. Most importantly, I would like to thank my Mom for teaching me how to write.

Contents

1	Introduction	13
1.1	Voting Games	13
1.2	Minimum Integer Weights	14
1.3	Notation	15
2	Prior Research	17
2.1	The Theory Behind Coalition Building	17
2.2	Empirical Evidence in Parliaments	19
2.3	Computation Difficulty of Finding the Minimum Integer Weights	21
2.4	Analogous Research in Neuroscience	21
3	The Algorithms	23
3.1	Coalition Enumeration	23
3.2	Interchangeable Parties	26
3.3	Party Rank Ordering	31
3.4	Game Definition Constraints	33
3.5	Coalition enumeration is NP-hard	33
3.6	Searching the Feasible Region for the Minimum Integer Weights	35
3.6.1	Revised Simplex	35
3.6.2	Dual Simplex with the Beale Tableau	36
3.7	Deriving Bound and Enumerate	39
3.7.1	Branch and Bound	39
3.7.2	Integer Branch and Bound	40

3.7.3	Bound and Enumerate	41
3.8	All-Integer Simplex	44
3.8.1	Finiteness of the Algorithm	45
3.8.2	Source Row Selection	46
3.8.3	Cut Formation and Deletion	47
3.8.4	All-Integer Simplex Example	47
3.9	Worst Case Orders of Growth of Algorithms	51
4	Results and Analysis	55
4.1	Methodology	55
4.2	Number of Coalitions and Constraints	57
4.3	Proportionality of Minimum Integer Weights	60
4.4	Homogeneity of Games	62
4.5	Running Times of Algorithms	63
4.6	Average Iterations for All-Integer Simplex	65
4.7	Solving an Arbitrary Game	66
5	Conclusions and Future Work	67
5.1	Conclusions	67
5.2	Improvements to the Algorithms	68
A	Technical Efficiencies, Optimizations, and Details	71

List of Figures

3-1	Top of a Branch and Bound Tree	39
3-2	Integer Branch and Bound	41
4-1	Incidence of Dictator Games by Number of Parties	56
4-2	Incidence of Dictator Games by Maximum Seats	56
4-3	Average Number of Min. Win. Coalitions by Number of Parties	57
4-4	Average Number of Min. Win. Coalitions by Maximum Seats	58
4-5	Average Number of Tying Coalitions by Maximum Seats	59
4-6	Composition of Simplex Rows Averaged over Number of Parties	60
4-7	Weight Share Regression Coefficients	61
4-8	Percent of Homogeneous Games by Number of Seats	62
4-9	Exponential Growth of All-Integer Simplex on Number of Constraints	65
4-10	Percent of Games Solved by the Three Algorithms	66

List of Tables

3.1	All Coalitions of Sample Game	24
3.2	Minimal Winning and Maximal Losing Coalitions of Sample Game . .	25
3.3	Minimal Winning and Unique Tying Coalitions of Sample Game . . .	26
3.4	Interchangeable Parties with Unequal Minimum Integer Weights . . .	31
3.5	Coalition and Rank Inequality Constraints Of Sample Game	34
3.6	Initial Beale Tableau for Weighted Voting Games	38
3.7	Example of Bound and Enumerate	43
4.1	OLS Regressions for Minimal Winning Coalitions and Simplex Rows .	60
4.2	Time Trials for the Three Algorithms	64

Chapter 1

Introduction

1.1 Voting Games

Integer programming techniques that find optimal solutions to constraint problems have far-reaching applications outside the realm of computer science and mathematics. I tailor the general approaches of all-integer simplex and branch and bound to the political science puzzle of finding the minimum integer weights of weighted voting games. In addition to giving scholars a research tool to quickly solve voting games, I show that minimum integer weights are not unique and confirm the theoretical result that weight share is proportional to *a priori* resources. After transforming branch and bound into a wholly new algorithm (bound and enumerate) and leaving all-integer simplex mostly as Gomory originally described, a third algorithm—augmented integer simplex—is developed by combining techniques from the other two methods.

Weighted voting games are scenarios in which players control certain numbers of votes, and in which some threshold of votes (usually a majority) is needed to distribute the spoils among the “winners.” The winners are those who contributed their votes to cross the threshold. Manifestations of voting games include fixed voting systems (e.g., European Union, Electoral College), elected parliaments (where the players are political parties), and shareholder blocs of corporations.

As long as one player does not have enough votes to single-handedly control the outcome, competing players must join together to form coalitions of votes. A winning

coalition is a group of players whose combined vote meets or exceeds the threshold. If a winning coalition would turn into a losing coalition if it lost any of its players, then it is called minimal winning. Enumerating these minimal winning coalitions is a key component of many operations on voting games.

1.2 Minimum Integer Weights

The minimum integer weights for a game assign a different set of votes (or weights) to the players of that game. Thus, the minimum integer weights also define a new voting game. The winning and losing coalitions for this new game must be the same as the original. As the designation “minimum integer” suggests, the new weights must all be integer values and there must be no smaller integer game that also produces the same coalitions. This minimality is helpful in determining whether augmenting one’s position with more votes will actually increase power [16]. If all minimal winning coalitions of a game have the same total minimum integer weight, then the game is homogeneous.

Stepping back to the larger problem of determining each player’s payoff (or power) in a game, notice that two subproblems are involved in the overall voting games. First, for a player to receive any pay-off, that player must be included in the winning coalition. Second, rarely is the distribution of pay-offs predefined; thus, the players in the governing coalition have differing levels of power, which affect the proportion of the spoils they can win for themselves. There are diverging branches of literature on these two topics. Only recently, in the works of Morelli [19] and Snyder [27], have scholars attempted to merge these two problems.

Minimum integer weights can be useful in solving both of the above problems. When forming a government, players must form a majority coalition that agrees on the government. An analogous setting is ministry allocation, where a majority (vote-wise) of coalition members must agree to the final government setup. The latter power relationship is more subjective, since portfolios have wide ranges of powers attached to them. On the other hand, coalition formation is binary in nature: a party

is either involved or not involved in the governing coalition. Because of this difference researchers used to focus solely on the question of government formation [6] [7]; more recently they have been tackling the harder problem of ministry allocation as well [30] [1].

1.3 Notation

A game is made of up a quota (or threshold) q and n players, each of whom starts with s_i seats (or weight). The notation for such a game is $[q; s_1, s_2, \dots, s_n]$. However, since this paper deals with parliaments and other majoritarian voting games, q is always a simple majority of the total weight. Thus, a “set of weights” will by itself determine a unique game. Coalitions are notated $\{p_1, p_2, \dots, p_i\}$, where each p represents a player, the “coalition size” is i , and the “coalition weight” is the sum of the weights for each player in the coalition.¹

¹Coalitions with parties that lack names will be designated as $[s_1, s_2, \dots, s_i]$. In this case, I will specify whether the combinations of congruent coalitions should be considered.

Chapter 2

Prior Research

2.1 The Theory Behind Coalition Building

William Gamson [11] was the first to examine the motivations and expectations of coalition players. He assumes that each player knows how much relative value the other players bring to the coalition, and uses Von Nuemann and Morgenstern's canonical work [29] to define the minimal winning coalition. Gamson theorizes that each player in the coalition will assert a payoff in proportion to the amount of resources contributed to the coalition. Hence, the resulting power of each player (if they are in the coalition) should be proportional to their beginning resources. This proportionality prediction has come to be known as "Gamson's Law."

Baron and Ferejohn [3] outlined a legislative bargaining process that contradicts Gamson's proportionality claim. Baron's model uses concepts from non-cooperative game theory and includes an ordering of proposal-makers. The general theory was developed for determining power in policy-making through the process of proposing, amending, and approving bills; however, Baron also applied his model to coalition formation. Baron argues that the proposal-maker, or *formateur* (in this case, the party that forms the coalition around it), has a disproportionately large amount of power.

The key difference between Baron's model and the proportional models are that Baron assigns different values to players whether they are the *formateur* or not. The

formateur's share is whatever is left over after paying the other parties their non-*formateur* values (or prices). Thus, this model is also dependent on how often each party is selected as *formateur*. Assuming that the probability a party is chosen as *formateur* is proportional to the minimum integer weights, the non-*formateur* Baron-Ferejohn values in odd, homogeneous games are always the minimum integer weight shares [27].¹

Morelli [19] disagreed with Baron's method, and described another type of non-cooperative framework for legislative bargaining. As in Baron's model, Morelli distinguishes between *formateurs* and non-*formateur*. But Morelli assigns higher demands to the non-*formateur* parties than Baron does. The result of Morelli's framework is that coalition payoffs are proportional to the number of seats each party has. Morelli's model (unlike Baron's) does not give any party higher status in three-player games with odd total weight, which makes sense from a minimum integer weight standpoint (since each party has a weight of one, no matter what the initial resources are).

Morelli [18] later extended his model by using the minimum integer weights as the demand ratios for the parties vying to be in the coalition. He uses a "demand bargaining set" to translate parties' demand ratios to the resultant coalition. Thus, Morelli was the first to combine coalition formation with payoffs from the resultant coalition. Given this combination, Morelli finds a proportional relationship between number of seats and "combined" power for homogeneous situations. He ignores heterogeneous games, which may partially explain the discrepancy between his results and other models.

Recently, Snyder, Ting, and Ansolabehere [27] have attempted to settle the multiple debates in the community. First, by using a theoretical analysis of infinite games, they find two types of game equilibria: (1) an "interior" equilibrium, for which power share is proportional to vote share, and (2) a "corner" equilibrium, for which power share for smaller parties is greater than what Gamson's Law predicts. Games with many "high weight" voters (or, put another way, where the quota is low compared to the largest parties) are more likely to have a corner equilibrium.

¹An example of an even, homogeneous game where this property does not hold is [5, 3, 3, 2, 1].

2.2 Empirical Evidence in Parliaments

Given the number of voting games that occur in practice, it would be folly to make theoretical predictions without verifying the result in the political arena. Eric Browne and Mark Franklin [6] noticed a lack of empirical data in the literature and set out to confirm Gamson's Law. Browne and Franklin operationalized a party's resources as the number of seats that party controls and the payoff as the number of cabinet positions given to the party. They then analyzed 13 parliaments from 1945 to 1969, resulting in 324 data points. Correlating payoff to seats yields a coefficient on seats of 1.07 (close to the ideal unity relation), and 86 percent of the payoff variance is explained purely by seats held. Browne and Franklin introduce the idea of "relative weakness" to explain why smaller parties get disproportionately higher payoffs in small governing coalitions (in terms of the number of coalition members).

Eric Browne revisited the question of the relationship between coalition payoff and seats in 1980, this time with John Frenreis [7]. They expanded the dataset of the previous study by eight years (obs=394) and corrected for "lumpiness" (incompatible denominators in total coalition votes and number of cabinet positions) and the relative weakness effect. The new analysis explained 93 percent of the variance of coalition payoff.

Two decades later, Paul Warwick and James Druckman [30] attempted to reconcile Browne and Franklin's findings of a proportional relationship with the series of theoretical works (starting with Baron's) that predict a disproportionate amount of power for the *formateur*. Warwick and Druckman compiled an extensive dataset, with 607 observations from 1945 to 1989. Their main contribution to this series of works is that they re-operationalized coalition payoffs. Realizing that some cabinet positions are more important than others, Warwick and Druckman weighted the portfolios, thus changing the dependent variable. Accounting for lumpiness and the relative weakness effect, they found a coefficient on seats of 0.987 (almost exactly proportionality).

However, Warwick and Druckman debunked Baron's model incorrectly. Instead

of using seat share as the independent variable, they should have used a proportional power index (such as the minimum integer weights). If such an index were used, the predictive abilities of both proportional and disproportional theoretical models could have been analyzed and contrasted. Instead, by using seat share, they must account for the relative weakness effect, thereby not actually predicting a proportional model. For instance, the game $[101; 100, 100, 1]$ has seat shares of approximately $[0.5, 0.5, 0.005]$, but the party with one seat has just as much power as the others. The minimum integer weights (along with any other reasonable index) predict the power share to be $[0.33, 0.33, 0.33]$. The algorithm presented in this paper gives researchers such as Warwick and Druckman the opportunity to use minimum integer weights in their analyses.

Recently, Ansolabehere et. al. [1] moved from the theoretical to the empirical side of the question. Using the algorithm presented in prior work [26] and in this paper, we compared European parliamentary data (1945-2000) to the predicted results of both the proportional and Baron-Ferejohn models. Baron-Ferejohn was the better fit in most cases, however the *formateur* effect was smaller than predicted. We also explored what factors predicated a party being chosen as the *formateur*. We found that minimum integer weight share, incumbency, and party rank have the greatest effect. Assuming only the first of these causes in theoretical models is thus a reasonable approximation.

Undoubtedly, as more data from across the globe is collected, further similar studies will be conducted. As the minimum integer weights are an integral part of both proportional and *formateur* models, an efficient algorithm to determine these weights is crucial. Past research enabled scholars to find the minimum integer weights for about 75% of large parliamentary games [26]; the algorithm detailed below fills this gap and allows researchers to solve almost any game.

2.3 Computation Difficulty of Finding the Minimum Integer Weights

Finding the minimum integer weights of a game relies on solving two independent, serial problems. First, the coalitions of the game must be enumerated to ensure that the solution weights form the same coalitions as the original game. Next, with the coalition constraints in hand, the minimum sequence of weights that satisfy those constraints must be found. Using a naive approach on the first problem is a reasonable solution. However, a reasonably efficient solution to the second problem can be elusive; thus, this paper is focused on the search subproblem.

Minimum integer weights are not the only way scholars measure voting power. Two of the most popular indices are Shapley-Shubik [24] and Banzhaf [2]. Yasuko Matsui and Tomomi Matsui [17] proved that calculating both Shapley-Shubik and Banzhaf values were NP-hard problems. They proved that both measures are NP-hard by reducing the each voting power problem to the well-known, NP-complete knapsack problem. This result directly leads to a proof (see Section 3.10) that coalition enumeration in general is NP-hard. Thus, the only way to find a polynomial-time algorithm for minimum integer weights would be to somehow avoid complete coalition enumeration. Future research will investigate this possibility.

The second subproblem of searching a constrained space for the minimum solution is an instance of integer linear programming. In general, integer linear programming (ILP) is NP-complete [23]. The most common technique for solving ILP problems is branch and bound [20]. I compare the efficiency of a variant of branch and bound, which I term bound and enumerate, to all-integer simplex [12], a lesser-known ILP technique. Both have exponential worst-case running times.

2.4 Analogous Research in Neuroscience

Led by Professor Jehoshua Bruck, a Caltech research group is developing efficient algorithms to model the neural networks found in brains; while this would seem

wholly unrelated to legislatures, the problems at hand are startlingly similar. Neural networks are the superposition of many linear threshold functions, which are functions that take in binary inputs and output a single binary number. This type of function is analogous to a coalition formation function that would take in a set of parties (or alternatively, a 1 if the party would be included in the coalition and 0 otherwise), and would output a 1 if the given coalition were winning and 0 if the coalition were losing. Such a coalition formation function would define a weighted voting game, and vice versa. Therefore, linear threshold functions also have minimum integer weights.

Previously, Bohossian and Bruck [5] demonstrated how to construct a set of integer weights that was guaranteed to be minimal. Transforming the weights to a threshold function is a simple process. Unfortunately, the inverse of this process does not allow one to take an arbitrary threshold function and subsequently determine the minimal weights. Bruck's research is currently attempting to solve that problem, which is also the main focus of this paper. While Bruck's team is using attributes of threshold function to guide the search for minimum integer solution, this paper focuses on using existing techniques in mathematics and computer science. Given both the success of this research and Bruck's group, future work might contrast the speeds of the two approaches.

Chapter 3

The Algorithms

The algorithms for finding minimum integer weights consist of two parts: (1) determining the defining constraints of a game, and (2) identifying the smallest sequence of integers that satisfies those constraints. Both of these subproblems are currently implemented in terms of NP-hard problems; thus, while I have refined the algorithms to increase efficiency, both grow exponentially with the size of the game.

3.1 Coalition Enumeration

Enumerating the coalitions of a voting game allows the game to be characterized by a set of inequalities. These inequalities will serve as the constraints for the minimization problem in the second half of the algorithm. The fewer redundant constraints used to define a problem, the more efficient the minimization problem will be; thus, the goal of the first subproblem will be to uniquely define a game in the fewest number of constraints possible.

A simplistic approach would be to mark all possible coalition combinations as “winning” or “losing,” depending on whether the total weight for the coalition is a majority. This approach yields exactly 2^n coalitions, where n is the number of parties in the game. The winning and losing coalitions for the sample game [5; 3, 2, 2, 1] are shown in Table 3.1.

For each coalition, an inequality is formed based on whether the coalition is win-

Game		Coalitions		
Party	Seats	Coalition	Seats	Status
<i>a</i>	3	\emptyset	0	Losing
<i>b</i>	2	$\{a\}$	3	Losing
<i>c</i>	2	$\{b\}$	2	Losing
<i>d</i>	1	$\{c\}$	2	Losing
Majority: 5		$\{d\}$	1	Losing
		$\{a, b\}$	5	Winning
		$\{a, c\}$	5	Winning
		$\{a, d\}$	4	Losing
		$\{b, c\}$	4	Losing
		$\{b, d\}$	3	Losing
		$\{c, d\}$	3	Losing
		$\{a, b, c\}$	7	Winning
		$\{a, b, d\}$	6	Winning
		$\{a, c, d\}$	6	Winning
		$\{b, c, d\}$	5	Winning
		$\{a, b, c, d\}$	8	Winning

Table 3.1: All Coalitions of Sample Game

ning or losing. Letting the seats of the i th party be s_i , and the majority (or quota) be q , then:

$$\sum_{i \in S} s_i \geq q \text{ where } S \text{ is a winning coalition of parties, and} \quad (3.1)$$

$$\sum_{i \in S} s_i < q \text{ where } S \text{ is a losing coalition of parties} \quad (3.2)$$

These 2^n inequalities define the weighted voting game. Any sequence of weights that meets these inequality constraints models the same game as the original sequence of seats.

Upon closer examination, this comprehensive approach to enumeration contains redundant information. First, we need not examine super-majority coalitions. In our example, since the coalition $\{a, b\}$ is winning, then clearly all coalitions that include, as a subset, the parties a and b must also be winning. The analogous argument is true for losing coalitions. From Table 3.1 we can safely remove the winning coalitions of $\{a, b, c\}$, $\{a, b, d\}$, $\{a, c, d\}$, and $\{a, b, c, d\}$, along with the losing coalitions of \emptyset , $\{a\}$,

$\{b\}$, $\{c\}$, and $\{d\}$, from our set of “defining” coalitions; thus, we also eliminate eight constraints. The only coalitions remaining are either minimal winning coalitions or “maximal losing coalitions” (i.e., adding any party would make it winning). These are enumerated in Table 3.2.

Game		Coalitions		
Party	Seats	Coalition	Seats	Status
a	3	$\{a, b\}$	5	Min. Winning
b	2	$\{a, c\}$	5	Min. Winning
c	2	$\{a, d\}$	4	Max. Losing
d	1	$\{b, c\}$	4	Max. Losing
Majority: 5		$\{b, d\}$	3	Max. Losing
		$\{c, d\}$	3	Max. Losing
		$\{b, c, d\}$	5	Min. Winning

Table 3.2: Minimal Winning and Maximal Losing Coalitions of Sample Game

A second redundancy can be eliminated through the rule of complementation. For each winning coalition, the coalition that includes all of the members not in the winning coalition must, by definition, be losing. (This only holds for majoritarian games.) Hence, we can eliminate all losing coalitions whose complements are winning from our defining inequalities. Note, however, that this reduction does not remove all losing coalitions. The complement of “tying coalitions”—losing coalitions whose sum is exactly half of the total sum—will not be winning either. But, in this case, we know its complement must be another tying coalition as well, so we need only one inequality for each pair of tying coalitions. This procedure changes our constraints slightly:

$$\sum_{i \in S} s_i \geq q \text{ where } S \text{ is a winning coalition of parties, and} \quad (3.3)$$

$$\sum_{i \in S} s_i = q - 1 \text{ where } S \text{ is a tying coalition of parties} \quad (3.4)$$

Implementing both these reductions results in only four “game defining” coalitions, listed in Table 3.3.

Game		Coalitions		
Party	Seats	Coalition	Seats	Status
a	3	$\{a, b\}$	5	Min. Winning
b	2	$\{a, c\}$	5	Min. Winning
c	2	$\{a, d\}$	4	Unique Tying
d	1	$\{b, c, d\}$	5	Max. Losing
Majority: 5				

Table 3.3: Minimal Winning and Unique Tying Coalitions of Sample Game

3.2 Interchangeable Parties

Given the minimal winning and unique tying set of inequality constraints, it would be possible to start searching for the smallest sequence of weights that is consistent with all the constraints. This approach would correctly find the minimum integer weights. However, more information can still be gleaned from the results of coalition enumeration. Helpful techniques for narrowing the potential search space include: (1) finding dummy players, (2) finding interchangeable parties, and (3) ordering the parties.

A “dummy player” is a party that is never included in a minimal winning coalition. These parties automatically are assigned the minimum integer weight of zero. Parties with at least one seat that are in unique tying coalitions will be included in a minimal winning coalition, so it is safe to ignore tying coalitions when searching for dummy players. (There are no dummy players in the given sample game.)

Some weighted voting games include “interchangeable parties.” Interchangeable parties are those that have the exact same power when forming coalitions, even though they might have a different number of a priori seats. In a more technical sense, parties a and b are interchangeable when (and only when):

$$\forall S. (1) S \text{ is minimal winning, } (2) a \in S, \text{ and } (3) b \notin S \Rightarrow (S - \{a\} + \{b\}) \text{ is also minimal winning.} \quad (3.5)$$

In the sample game, parties b and c are clearly interchangeable since they have the same number of seats. The following theorem demonstrates how to find inter-

changeable parties that do not have the same number of seats.

Theorem 3.1. *Parties a and b are interchangeable if and only if for each coalition size, they are included in the same number of minimal winning coalitions of that size.*

Proof. The “only if” direction is trivial. By the definition of interchangeable, for all minimal winning coalitions that a is in, then b must either be in that same coalition or in an analogous coalition of the same size (namely, $S - \{a\} + \{b\}$). The same is true for all coalitions that include b . Thus, a and b will form the same number of coalitions for each coalition size.

Instead of proving the “if” direction directly, I will prove the contrapositive: If a and b are not interchangeable, then there exists a coalition size for which a and b belong to different numbers of coalitions of that size. Assume, without loss of generality, that there is some coalition of size k that includes a but not b and that swapping b in for a would produce a losing coalition. Thus, b has a lower weight than a . For all coalitions of size k that include b but not a , swapping a in for b would produce either (1) a minimal winning coalition of size k or (2) a superset of a minimal winning coalition of size less than k (let this size be j) that includes party a . If, when swapping all of b 's k -sized coalitions, only situation 1 results, then a must be in more k -sized minimal winning coalitions than b . If situation 2 results, then repeat this procedure for size j , as our initial assumption would still hold for j -sized coalitions. When $k = 1$, situation 2 cannot occur, and the procedure terminates. Thus, there must exist a coalition size for which a belongs to more minimal winning coalitions than b .

Note that only minimal winning coalitions, and not tying coalitions, are relevant. If party a is involved in a tying coalition and (1) the coalition does not include b , and (2) the analogous coalition including b and excluding a does not exist, then swapping a and b would necessitate a minimal winning coalition that a is in and b is not in (or vice versa). In essence, if there is a mismatch in power in the tying coalitions, this mismatch will manifest itself in the minimal winning coalitions as well. \square

This characteristic allows for an easy way to find interchangeable parties and order

such interchangeable sets. A set of interchangeable parties that is included in more k -sized coalitions must have a larger weight than the interchangeable parties that are in fewer coalitions of size k .¹ But, must all interchangeable parties have the same minimum weight? What about minimum *integer* weight? Intuition tends toward affirmative answers, but only the former is true.

First note that minimum weights do not actually exist. Given a game, one can divide the weights by an arbitrarily large number to get arbitrarily small weights. These new weights, being proportional to the original weights, will still define the same game. Thus, for logic's sake, let the sequence of minimum weights be the smallest weights that define the original game and that has a weight of 1 as its smallest weight.

Lemma 3.2. *Given a game with interchangeable parties a and b , which have respective weights w_a and w_b , if $w_a \neq w_b$, then there exists a smaller set of weights that (1) defines the same game and (2) does not decrease the smallest weight.*

Proof. Let the smaller, new set of weights be the same as the original weights but with w'_a and w'_b as the weights for parties a and b instead of w_a and w_b . Assume, without loss of generality, that $w_a > w_b$. These new weights are defined as:

$$\begin{aligned}
 w'_a &= w_a - \gamma - \epsilon, \text{ where } \epsilon \text{ is an appropriately small positive number.} \\
 w'_b &= w_b + \gamma - \epsilon \\
 w'_i &= w_i, \text{ for } i = 1, 2, \dots, n \text{ (} i \neq a \wedge i \neq b \text{)} \\
 \gamma &= \frac{1}{2}(w_a - w_b)
 \end{aligned} \tag{3.6}$$

These new sets of weights are smaller than the original by $2 * \epsilon$. To show that the new weights still define the same game, I must show that all the original winning coalitions are still winning. There are three cases of winning coalitions. First, consider winning coalitions that do not include either parties a or b . Since $w'_a + w'_b < w_a + w_b$, the winning coalitions increase their margins of victory in the new game. Next, consider winning coalitions that include either party a or party b , but not both.

¹Further clarification of and discussion on this point is in Section 3.3.

Since a and b are interchangeable, these types of winning coalitions come in pairs. Of the pair, the winning coalition that includes b is clearly still winning since $w'_b > w_b$ and $w'_a < w_a$. Since, in the new game, $w'_a = w'_b$, both coalitions in the pair have the same weight; thus, both are winning. Finally, consider coalitions that include both parties a and b . Each of those coalitions must have a greater weight than its complement; denote this difference as δ . If ϵ is chosen such that, for every δ , $2 * \epsilon < \delta$, then the coalitions will still be winning even with the new, smaller weights. Thus, the new weights define the same game.

For the second proof requirement, clearly the smallest weight does not decrease. The only weight to decrease is w_a , which cannot be the smallest weight since I assume $w_a > w_b$. □

Lemma 3.3. *Interchangeable parties have the same minimum weight.*

Proof. By contradiction. Assume there is a sequence of minimum weights for which interchangeable parties a and b do not have the value. Then by Lemma 3.2, there exists a smaller set of weights that defines the same game with smallest weight greater than or equal to one. If the smallest weight is equal to one, then these new weights are minimum. If the smallest weight is greater than one, then scaling all the weights down so that the smallest weight is equal to one results in minimum weights. In either case, the original weights are not minimum. □

The above proofs indicate that having interchangeable parties with different weights is inefficient. Therefore, can one assume that interchangeable parties will have the same minimum integer weight, thus reducing the number of variables needed to solve the voting game problem? First I attempt to use the same proof technique used in Lemma 3.2 to show that interchangeable parties have the same minimum integer weight.

Proposition 3.4. *Interchangeable parties have the same minimum integer weight.*

Proof Attempt. Assume (similarly to Lemma 3.2) that w_i are integer weights, a and b are interchangeable, and $w_a > w_b$. To end with a smaller integer set of weights,

the following transformations would need to be done:

$$\begin{aligned}
w'_a &= w_a - 1 \\
w'_b &= w_b \\
w'_i &= w_i, \text{ for } i = 1, 2, \dots, n \text{ (} i \neq a \wedge i \neq b \text{)}
\end{aligned} \tag{3.7}$$

Note the differences between (3.7) and (3.6). To decrease the size of the game, one needs to decrease a party's weight by at least one, not some arbitrarily small value. That restriction becomes the Achilles' heel of the proof.

As before, when determining whether these new weights define the same game, there are three types of coalitions to consider. First, coalitions that include neither parties a nor b are still winning, since $w'_a + w'_b < w_a + w_b$. Second, consider coalitions that include either parties a or b , but not both. Using the logic of Lemma 3.2, the coalition that includes b and not a is still winning, since $w'_b = w_b$ and $w'_a < w_a$. The second coalition in the pair, which includes a but not b , remains winning since (1) $w'_a \geq w'_b$ and (2) the first coalition in the pair is winning.

The proof fails on the third type of coalition, which has both parties a and b as members. If the coalition won by one vote with the original set of weights, then since $w'_a + w'_b = w_a + w_b - 1$, the coalition would be losing in the new scheme. The fact that the transformation given above does not work in this case by no means indicates that a smaller set of integer weights (with $w'_a = w'_b$) does not exist. However, this smaller set is also not guaranteed to exist.

Theorem 3.5. *Minimum integer weights are not unique.*

Proof. The fact that the above proposition cannot be proven begs the question of whether there exists a counterexample to the claim. After searching thousands of randomly generated 12-player games, a case in which interchangeable parties do not share the same weight was found. In games such as the one detailed in Table 3.4, there are multiple possible minimum integer weight assignments. The “constrained integer” row shows the result when interchangeable parties are constrained to have the same value. In the example, another valid minimum integer weight scheme would

	Parties												Total
	1	2	3	4	5	6	7	8	9	10	11	12	
Original	56	53	47	46	37	36	30	10	6	6	4	3	334
Minimum Integer	55	52	46	45	36	35	30	10	6	6	3	4	328
Constrained Integer	61	58	51	50	40	39	34	11	7	7	4	4	366

Parties 11 and 12 are interchangeable

Table 3.4: Interchangeable Parties with Unequal Minimum Integer Weights

have party 11 with a weight of four and party 12 with a weight of three. Thus, minimum integer weights are not unique. \square

Corollary 3.6. *Minimum integer weights are not necessarily monotonic.*

Proof. Table 3.4 gives an example of a sequence of minimum integer weights that, when ordered by seat share, are not monotonic. \square

3.3 Party Rank Ordering

The fact that interchangeable parties may have different minimum integer weights is an unfortunate blow to the efficiency of the algorithm, as now one variable must be used for each party. However, it would still be beneficial to give a partial ordering to the parties. Thus, the goal is to assign ranks to the parties.

Definition Let n parties (excluding dummy players) be assigned ranks $1, \dots, r$. Let w_i be the minimum integer weight of the i th party. The following characteristics hold:

- 1) $r \leq n$
- 2) $\forall i, j. \text{rank}(i) < \text{rank}(j) \Rightarrow w_i > w_j$ (3.8)
- 3) $\forall i. \text{rank}(i) = 1 \Rightarrow w_i \geq 1$

Lemma 3.7. *Non-interchangeable parties have different minimum integer weights.*

Proof. By contradiction. Given a set of minimum integer weights, any two parties with the same weight must be interchangeable since they would clearly meet the

requirements of 3.5. Therefore, non-interchangeable parties must have different minimum integer weights. \square

Thus, assigning parties to the same group if and only if they are interchangeable will result in party rank. Property (1) of (3.8) holds since there cannot be more groups than parties. Property (3) holds since all minimum integer weights (for non-dummy parties) must be greater than or equal to one. Property (2) holds from Lemma 3.7, but one might not necessarily be able to order these groups before discovering the minimum integer weights.

Theorem 3.8. *Parties can be divided into ranks in based solely on a priori knowledge of coalition formation.*

Proof. First, as suggested above, collect all parties into groups based on interchangeability. Theorem 3.1 proves useful in this endeavor. Starting with $k = 1$, assign the higher rank (a rank of one being considered “higher” than a rank of two) to groups of parties that are involved in the greater number of k -sized minimal winning coalitions. Break ties by incrementing k and repeating.

Assume party a is included in more k -sized minimal winning coalitions than party b . Thus, there exists one of these coalition that includes a and not b . Adding b to the coalition would either (1) produce a losing coalition, or (2) produce a superset of a j -sized minimal winning coalition ($j < k$). In situation (1), party a has a larger weight than party b are the procedure correctly assigns a the higher rank. In situation (2), party b has a larger weight, and since the procedure checks j -sized coalitions before k -sized, party b will appropriately receive the higher rank.

Thus, parties can be ranked by only using information available from coalition formation. \square

Corollary 3.9. *For each game, there exists a monotonic sequence of minimum integer weights.*

Proof. Using the same logic as the proof in Theorem 3.8, one can see that if party a has a higher rank than party b , then party a also has a higher seat share. Thus,

ordering the parties by seat share will also order the parties by weight share with the exception of within groups of interchangeable parties. However, since in these groups the weights can be distributed in an arbitrary manner (the parties are interchangeable after all), then by monotonically ordering the weights within each rank, a monotonic weight sequence is produced. \square

3.4 Game Definition Constraints

Ranks are useful in that they provide a more compact way to define a game. By using ranks instead of party values for variables in coalition enumeration, the number of coalitions produced is significantly reduced. In the sample game presented in Table 3.3, the coalitions $\{a, b\}$ and $\{a, c\}$ are now both $[1, 2]$ as a 's rank is 1 and the rank of both b and c is 2. (Note that coalitions are no longer sets since they can contain duplicate values.) I refer to this smaller set of coalitions as “unique rank coalitions;” the same principle can be applied to tying coalitions as well.

While this reduction helps save space in some areas of research (e.g., Baron-Ferejohn values), because of the failure of Proposition 3.4, the constraint variables must be the parties themselves. The constraints used to find the minimum integer weights are that: (1) the minimal winning coalitions must win by at least one “vote,” (2) the unique tying coalitions must tie, (3) each party’s weight must be at least one greater than the weight of a party one rank below the bigger party, and (4) the parties with the smallest rank must be at least one. Table 3.5 lists the eight constraints that would be used to find the minimum integer weights of the sample game. In general, the number of constraints is the sum of the number of minimal winning coalitions, unique tying coalitions, and ranks.

3.5 Coalition enumeration is NP-hard

To prove that coalition enumeration is NP-hard I reduce the known NP-complete problem of calculating Banzhaf power indices [17] to coalition enumeration. The

Game			
Party	Seats	Rank	Resulting Inequality
a	3	1	$w_a - w_b \geq 1$
b	2	2	$w_b - w_d \geq 1$
c	2	2	$w_c - w_d \geq 1$
d	1	3	$w_d \geq 1$
----- Majority: 5 -----			
Coalitions			
Coalition	Seats	Status	Resulting Inequality
$\{a, b\}$	5	Min. Winning	$w_a + w_b - w_c - w_d \geq 1$
$\{a, c\}$	5	Min. Winning	$w_a - w_b + w_c - w_d \geq 1$
$\{a, d\}$	4	Unique Tying	$w_a - w_b - w_c + w_d = 0$
$\{b, c, d\}$	5	Min. Winning	$-w_a + w_b + w_c + w_d \geq 1$

Table 3.5: Coalition and Rank Inequality Constraints Of Sample Game

input to Banzhaf is the same as the input to coalition enumeration—namely, a weighted voting game. A party’s Banzhaf score is proportional to the number of coalitions for which that party is a “swing” voter (i.e., without that party, the coalition would be losing). The Banzhaf value is normalized so that the sum of the scores is unity. Note that more coalitions are involved in this calculation than just the minimal winning coalitions, since minimal winning coalitions are composed of *only* swing voters, while “Banzhaf” coalitions need only *one* swing voter.

Theorem 3.10. *Coalition Enumeration is NP-hard.*

Proof. I solve the NP-hard Banzhaf problem in terms of coalition enumeration. The first step in solving Banzhaf would be to list the all minimal winning coalitions. Then, for each party in each coalition, the numerator of that party’s Banzhaf score (as well as the common Banzhaf denominator) would be incremented by one plus the number of parties outside the coalition that, if added to the current coalition, would keep the current party swing.² Since the running time of coalition enumeration is already a function of the number of coalitions times the number of parties, this extra procedure only multiplies a factor of n to that running time. Thus, coalition enumeration cannot

²This procedure, in effect, enumerates all the Banzhaf coalitions. Care must be taken not to repeat coalitions.

be easier to solve than Banzhaf and is thus NP-hard. □

Given this result, I am unconcerned that the current coalition enumeration algorithm runs in time $O(2^n)$. The current algorithm is just a depth-first, British-museum search on a binary tree. Each node is a party, and the children are whether to add that party to the coalition or not. Thus, no party is repeated in the coalition. If adding a party results in a tying coalition, that coalition is recorded as such. If adding a party results in a winning coalition, then the coalition is recorded and that sub-tree is pruned. If the nodes are ordered so that the largest party is decided first and the smallest party last, then all recorded winning coalitions will also be minimal winning. The depth of the tree is n ; thus, the running time is $O(2^n)$. In Section 4.2 I show that the number of coalitions grow exponentially with the size of the game, thus a much more thoughtful approach is needed to find some sort of polynomial algorithm. Potential avenues of future research are also discussed in that section.

3.6 Searching the Feasible Region for the Minimum Integer Weights

With a set of constraints, the problem of finding the minimum integer weights reduces to an integer linear programming (ILP) problem. A common technique for solving ILP problems is called branch and bound; and a variant of this method, termed bound and enumerate, is developed in this paper. A separate, less-common technique of all-integer simplex is also tried and found to be, on average, faster than bound and enumerate. Both bound and enumerate and all-integer simplex use the well-known linear programming (LP) simplex algorithm at some level.

3.6.1 Revised Simplex

The simplex algorithm takes a set of constraints and a cost function and returns the sequence of variables that satisfies all constraints and has minimum cost. Since, in this case, the variables are the weights of each party, the cost function is simply the

sum of the weights. Simplex finds the optimal point by moving from extreme point to extreme point in the n -dimensional feasible search space. The algorithm uses the cost function to choose the best constraint to move along and thereby finding the next extreme point. There are many versions of the simplex algorithm; the first tried was the two-phase revised simplex method detailed by Best [4].

Revised simplex first converts inequalities to equalities by adding “slack variables.” These variables absorb the difference between the two sides of the inequality. To ensure that the algorithm starts with a feasible point, an artificial optimization problem (called “phase one”) is performed. Another set of variables, “artificial variables,” are added to every equality and \geq inequality constraint.

With the addition of slack and artificial variables, there are more variables than equations. In a linear programming problem, however, the number of non-zero variables cannot be the number of constraints. Thus, variables are separated into two categories: basic and non-basic. Basic variables are allowed to be non-zero, and the set of basic variables is called the “basis.”

Revised simplex starts with all the artificial variables and the slack variables of the \leq inequalities as the basic variables. An artificial cost function ensures that at the termination of phase one all artificial variables are removed from the basis.³ The party and slack variables that took their places during phase one start in the basis for phase two, during which the original cost function is optimized. If a feasible solution exists, and for voting games one must exist, all the party variables move into the basis. The weights of the parties are easily gleaned from terminating information (i.e., the final matrices).

3.6.2 Dual Simplex with the Beale Tableau

The two-phase revised simplex algorithm has large inefficiencies in terms of both time and space. With regard to running time, a dual approach allows for an infeasible

³Occasionally, an artificial variable will remain in the basis; its value is guaranteed to be zero. Unfortunately, this situation indicates a redundant constraint and the phase one must be re-run with the constraint removed.

starting point, thus avoiding the necessity for two phases. The minimum integer weight problem will always have an infeasible starting point since the origin (i.e., $\forall i. w_i = 0$) does not define a voting game. The dual problem starts at the origin and systematically increases the weights until a feasible is found. The first sequence of weights in the “primal” feasible region is the solution to the problem.

For technical reasons, the dual method handles only \leq inequality or equality constraints. Thus, three of the constraint types must be scaled by negative one. The resulting equations that define a game are

$$\sum_{i \in S} -w_i + \sum_{i \notin S} w_i \leq -1 \quad \text{where } S \text{ is a winning coalition of parties} \quad (3.9)$$

$$\sum_{i \in S} w_i - \sum_{i \notin S} w_i = 0 \quad \text{where } S \text{ is a unique tying coalition} \quad (3.10)$$

$$\forall i. \begin{cases} -w_i + w_j \leq -1 & \text{where } \text{rank}(i) = \text{rank}(j) + 1 & \text{rank}(i) > 1; \\ -w_i \leq -1 & & \text{rank}(i) = 1. \end{cases} \quad (3.11)$$

Tangentially, the Beale tableau is a more efficient way of keeping track of the basic and non-basic variables (as opposed to the canonical form) [22].⁴ Given a game with m constraints and n variables of the form

$$\min. \quad z = \sum_{j=1}^n 1 \quad (3.12)$$

$$\forall 1 \leq i \leq m. \quad \sum_{j=1}^n a_{i,j}(x_j) \leq b_i \quad (3.13)$$

the initial Beale tableau is shown in Table 3.6 [22]. The advantages of the Beale tableau are that no matrix inversion is needed and that the information on basic and nonbasic variable is kept in one place. Pivoting⁵ on the Beale tableau is slightly

⁴Implementations of the simplex algorithm usually use a “tableau” of numbers to hold the data through the iterations. An example of a tableau, for those unfamiliar, is given later in this section. “Canonical form” includes the original constraints, plus all the surplus and artificial variables. Revised simplex splits the tableau into separate matrices; the two main matrices hold the coefficients of the basic and non-basic variables respectively.

⁵Pivoting is a method to move a variable into or out of the basis. Pivoting in canonical form involves scaling each row by a multiple of a specific “pivot row” in such a way to make all elements

All Variables	Constant Values	Nonbasic variables			
		$(-x_1)$	$(-x_2)$	\cdots	$(-x_n)$
$(-z)$	$(-z)$	1	1	\cdots	1
x_1	0	-1	0	\cdots	0
x_2	0	0	-1	\cdots	0
\vdots	\vdots	\vdots	\vdots		\vdots
x_n	0	0	0	\cdots	-1
x_{n+1}	b_1	$a_{1,1}$	$a_{1,2}$	\cdots	$a_{1,n}$
\vdots	\vdots	\vdots	\vdots		\vdots
x_{n+m}	b_m	$a_{m,1}$	$a_{m,2}$	\cdots	$a_{m,n}$

Table 3.6: Initial Beale Tableau for Weighted Voting Games

irregular in that: 1) the pivot column (including the pivot element) is scaled by the negative inverse of the pivot element, 2) the general pivot is done based on columns, thus the pivot row (excluding the pivot element) will be set to zero. Examples of pivoting are shown in Section 3.8.4.

The one disadvantage of the Beale tableau is that equalities can no longer be expressed. Thus, tying coalitions must be in pairs; both pairs of coalitions are constrained to be ≤ 0 . That work-around forces both coalitions to be 0. Unique tying coalitions are therefore no longer pertinent.

A “constant value” column with no negative values indicates an optimal solution and the process terminates. Until that point, a pivot row and pivot column need to be chosen at each iteration. The pivot row is simply the row (excluding the cost row) with the most negative constant value. Choosing the pivot column is more complex. Letting the pivot row be r and each tableau element be represented by $\bar{a}_{i,j}$, the pivot column is the lexicographically smallest⁶ when columns are scaled by their respective $|\bar{a}_{r,j}|$ and after excluding the “constant value” column and any column with $\bar{a}_{r,j} \geq 0$.

of the “pivot column” zero, with the exception of the intersection of the pivot row and pivot column, which becomes one.

⁶Precedence is given to the top of the tableau—cost row included. Formally, column p is lexicographically smaller than column q when the first term of $\bar{a}_{i,p} - \bar{a}_{i,q}$ is negative. Ties are avoided since the $n \times n$ identity matrix at the top of the initial tableau ensures that all columns are linearly independent.

3.7 Deriving Bound and Enumerate

3.7.1 Branch and Bound

While the set of weights calculated by the simplex algorithm is guaranteed to be minimal, the weights are not guaranteed to be integral. For odd, homogeneous games with no tying coalitions, however, the weights actually are always integral. For all other games, some sort of integer programming algorithm is needed. Branch and bound, if given a large amount time and resources, will do the trick.

Branch and bound starts by running the simplex algorithm on the initial problem. This problem is treated as the root of a search tree. If there are no non-integer weights in the solution, then the algorithm stops—those weights are the minimum integer weights. If there are non-integer weights, then for each of those weights, two additional simplex problem are produced: one in which the offending variable is constrained to be at most the floor of its current value, and another in which the variable is constrained to be at least the ceiling of its current value. Thus, if the root solution has two non-integer weights, the root node will have four children in the graph (see Figure 3-1). Each subproblem is put in a queue, which is ordered by

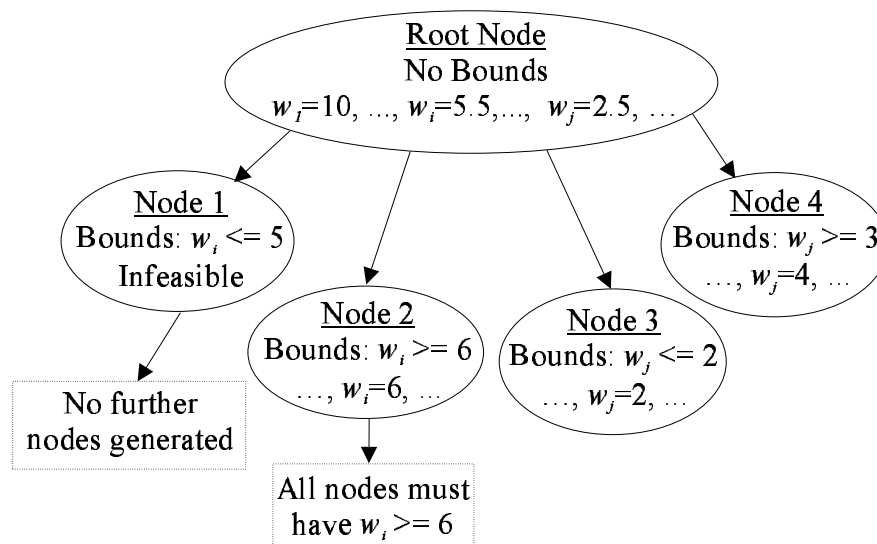


Figure 3-1: Top of a Branch and Bound Tree

ascending cost.

This ordering implies that if an integer solution is found, it must be minimal, since all subproblems of least cost have already been analyzed. The key notion is that the costs of the children of a node must be at least the cost of the parent node. Children are formed by adding constraints; since additional constraints can only reduce the search space, the solution to a child node cannot be more efficient than the solution to the parent. Many subproblems will be infeasible (especially nodes with upper-bounded variables); these nodes are not included in the queue, which effectively prunes their sub-trees.

3.7.2 Integer Branch and Bound

A key characteristic of the voting game problem is that the coefficients of the cost function are all integers. Thus, the total cost must be integral. However, branch and bound searches many nodes that have non-integer total cost. Watching branch and bound visit over 10,000 nodes in games with as few as 10 players is painful, since the vast majority of those nodes could be ruled out since they have non-integer cost.

While these nodes can be ruled out after the LP relaxation problem has been executed and their total cost has been found, the puzzle of avoiding the nodes before their cost is known is trickier. One solution is to add a constraint to the LP problem restricting the total cost to be a specific integer value. Instead of minimizing cost, one could minimize the weight of the largest party.

To find which values to restrict cost to, first, just as in branch and bound, run simplex (minimizing total cost) on the root node. The ceiling of the total cost of the root node gives a lower bound for the total cost of the minimum integer weights. For example, if the cost of the root is 252.63, then the minimum integer weights must sum to at least 253; if the sum was 252 or below, then the root LP problem would have found that solution.

Start with k as the ceiling of the root's cost. Run branch and bound with the additional constraint the total cost = k . If an integer solution is found, those values are the minimum integer weights. If branch and bound terminates with no solution,

increment k and repeat. Figure 3-2 illustrates this concept.

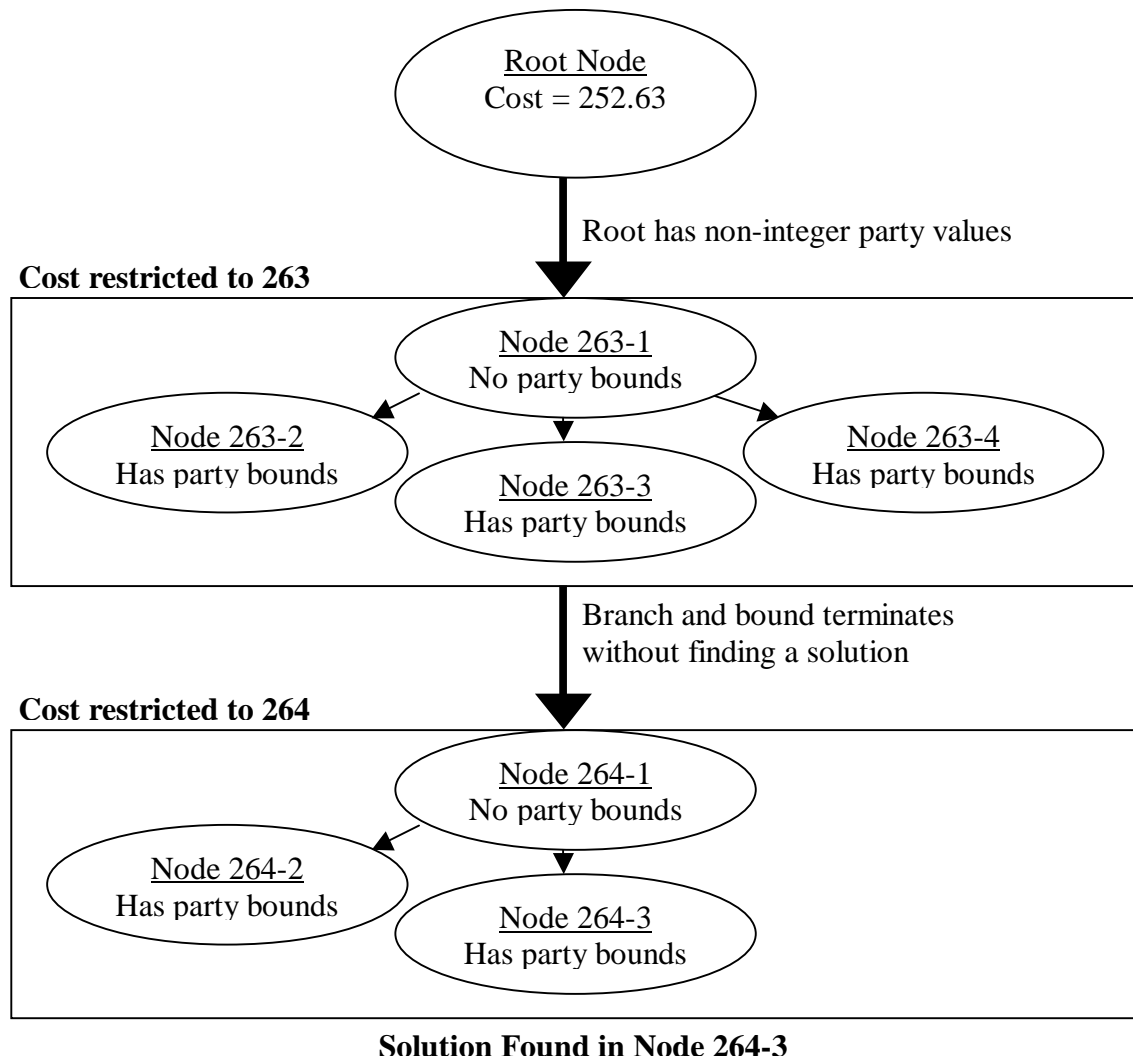


Figure 3-2: Integer Branch and Bound

3.7.3 Bound and Enumerate

Even with the cost restriction, one iteration of integer branch and bound can easily explode exponentially. For each iteration, a simpler subproblem would be whether there exists an integer solution with that specific cost. Unfortunately, characteristics of empty lattice spaces is an “open question” [21]. Advances in this topic would make variants of integer branch and bound much more efficient by allowing the algorithm to skip over certain cost levels.

Letting the true scholars tackle the general problem of empty lattice spaces, there is some basic logic that can shed light on whether a space has an integer point in it. Consider restricting total cost to be k and minimizing w_i . The root node of the k iteration of branch and bound might return a non-integer solution with $w_i = 22.3$. Running this same node again, but now maximizing w_i , returns a solution with $w_i = 22.8$. Clearly, no integer solution can be present with total cost being k , because no integer value is available for w_i .

Expanding this concept to the entire game, at each cost level k each weight variable is minimized and maximized. Let the minimum and maximum values for w_i at cost level k be μ_i^k and ν_i^k , respectively. If, for any variable i , $\lceil \mu_i^k \rceil > \lfloor \nu_i^k \rfloor$, then there is no integer solution with cost k . Note that μ_i^k and ν_i^k will always exist, since the original sequence of seats can be scaled by $(k / \sum s_i)$ to produce weights that both sum to k and define the same game.

Running $2n$ LP relaxation problems at each cost level significantly reduces the potential search space and avoids the necessity of executing branch and bound. Instead, given the lower and upper bounds for each variable ($\lceil \mu_i^k \rceil$ and $\lfloor \nu_i^k \rfloor$), enumerate all possible combinations of integer values that satisfy those bounds. Simply check whether an individual combination satisfies the minimal winning and unique tying constraints; if the sequence of weight satisfies all conditions, these are the minimum integer weights.

Additional restrictions can be placed on the possible combinations. First, the sum of the variables should be k ; otherwise, combinations will be repeated over iterations, and minimality of the resulting weights is not guaranteed. Second, Corollary 3.9 proves the existence of a monotonic solution, thus one may safely ignore combinations that violate the rule $\forall i, j. s_i \leq s_j \Rightarrow w_i \leq w_j$. Both restrictions cut down on the number of combinations tested while still guaranteeing that a solution will be found.

Table 3.7 illustrates the branch and enumerate algorithm executed on the game [104; 71, 53, 31, 21, 11, 9, 7, 3]. The algorithm, after running one LP relaxation problem, jumps immediately to a cost of 48. Nine iterations later, the algorithm enu-

	Parties							
	1	2	3	4	5	6	7	8
Seats (s_i)	71	53	31	21	11	9	7	3
Root node determines minimum cost of 48								
Iteration 1, Cost = 48								
Minimum ($\lceil \mu_i^{48} \rceil$)	<i>15</i>	10	9	6	<i>4</i>	<i>3</i>	<i>2</i>	<i>2</i>
Maximum ($\lfloor \nu_i^{48} \rfloor$)	<i>14</i>	10	8	6	<i>3</i>	<i>2</i>	<i>1</i>	<i>1</i>
Enumerated Combinations: 0								
Iteration 2, Cost = 49								
Minimum ($\lceil \mu_i^{49} \rceil$)	15	10	9	6	<i>4</i>	<i>3</i>	<i>2</i>	<i>2</i>
Maximum ($\lfloor \nu_i^{49} \rfloor$)	15	10	9	6	<i>3</i>	<i>2</i>	<i>1</i>	<i>1</i>
Enumerated Combinations: 0								
⋮								
Iteration 8, Cost = 55								
Minimum ($\lceil \mu_i^{55} \rceil$)	17	12	9	6	4	3	<i>2</i>	<i>2</i>
Maximum ($\lfloor \nu_i^{55} \rfloor$)	18	13	10	7	4	3	<i>1</i>	<i>1</i>
Enumerated Combinations: 0								
Iteration 9, Cost = 56								
Minimum ($\lceil \mu_i^{56} \rceil$)	17	12	9	6	4	3	<i>2</i>	<i>2</i>
Maximum ($\lfloor \nu_i^{56} \rfloor$)	18	14	10	8	4	3	<i>2</i>	<i>2</i>
Enumerated Combinations: 4								
Iteration 10, Cost = 57								
Minimum ($\lceil \mu_i^{57} \rceil$)	17	12	9	6	4	3	<i>2</i>	<i>2</i>
Maximum ($\lfloor \nu_i^{57} \rfloor$)	19	14	11	8	4	3	<i>2</i>	<i>2</i>
Enumerated Combinations: 10								
⋮								
Iteration 23, Cost = 70								
Minimum ($\lceil \mu_i^{70} \rceil$)	20	14	9	6	4	3	<i>2</i>	<i>2</i>
Maximum ($\lfloor \nu_i^{70} \rfloor$)	25	21	14	11	5	4	<i>2</i>	<i>2</i>
Enumerated Combinations: 71								
Iteration 24, Cost = 71								
Minimum ($\lceil \mu_i^{71} \rceil$)	21	15	9	6	4	3	<i>2</i>	<i>2</i>
Maximum ($\lfloor \nu_i^{71} \rfloor$)	26	21	14	11	5	4	<i>2</i>	<i>2</i>
Enumerated Combinations: 471								
Iteration 25, Cost = 72								
Minimum ($\lceil \mu_i^{72} \rceil$)	21	15	9	6	4	3	<i>2</i>	<i>2</i>
Maximum ($\lfloor \nu_i^{72} \rfloor$)	26	22	14	12	5	4	<i>3</i>	<i>3</i>
Solution (w_i)	22	15	13	9	5	4	<i>2</i>	<i>2</i>

Weight constraints that eliminate all possible combinations are *italicized*.

Table 3.7: Example of Bound and Enumerate

merates and tests its first possible weight sequences. The valid search space grows exponentially with additional iterations; this explosion is the fatal weakness of the algorithm. In the example, subtle changes in three values between iterations 23 and 24 lead to a 660% increase in the number of sequences to examine.

3.8 All-Integer Simplex

In the early 1960's, Ralph Gomory [12] [14] developed a variant of the simplex algorithm in which all tableau entries are kept as integers. The theory behind the algorithm is that instead of moving along a constraint to the next extreme point, as one does in normal simplex, a constraint (or "cut") is added to the system which reduces the feasible region while creating an alternative extreme point [13] [10].

As with normal simplex, the voting weight problem is easier to solve via a dual method. The exact algorithm used is detailed by Salkin and Mathur [22]; little changed from the process originally introduced by Gomory in 1963 [12]. Again, the Beale tableau is utilized, but in this case all the elements remain integers. From (3.9) and Table 3.6, the original tableau only consists of integers. Thus, if the additional cuts are all-integer rows, and the ensuing pivot keeps the columns integral, then the entire tableau (and hence the solution) will never contain a fractional value.

The rules for adding a cut are as follows. First, choose the pivot row (now designated the "source row," s) as described in Section 3.6.2. Then add the next pivot row x_r row to the tableau with the following equation

$$x_r = \lfloor \frac{\bar{a}_{s,0}}{\lambda} \rfloor + \sum_{j=1}^n \lfloor \frac{\bar{a}_{s,j}}{\lambda} \rfloor (-x_j) \geq 0 \quad (3.14)$$

Pivot with row r and column p ; if the new tableau is suboptimal proceed to the next iteration.

The only variable unspecified in the above cut equation is λ . The rules for choosing λ (and pivot column p) are a bit complex and are as follows.

1. Given a source row s , exclude from consideration the "constant value" column

and columns with $\bar{a}_{s,j} \geq 0$.

2. Choose pivot column p as the lexicographically smallest of the eligible columns.
3. Excluding column p , let u_j be the largest integer ≥ 1 such that column j scaled by the inverse of u_j is still lexicographically larger than column p . Set $u_p = 1$. If no such u_j for column j exists (e.g., $\forall i < y. \bar{a}_{i,j} = \bar{a}_{i,p} = 0$; $\bar{a}_{y,j} > 0$ and $\bar{a}_{y,p} < 0$), then set u_j to an arbitrarily large value.
4. Set $\lambda_j = -\bar{a}_{s,j}/u_j$.
5. Use the maximum of all valid λ_j 's as the λ for the cut and its column as the pivot column.

This process produces $\lambda \geq 1$ since $\lambda \geq \lambda_p = -\bar{a}_{s,p}/u_p = -\bar{a}_{s,p} \geq 1$. Also note that $\lambda \geq -\bar{a}_{s,p}$; thus, the pivot element term in (3.14), $\lfloor \frac{\bar{a}_{s,p}}{\lambda} \rfloor$, is always -1. A pivot element of -1 will keep all tableau entries as integers, thus fulfilling the main characteristic of all-integer simplex.

3.8.1 Finiteness of the Algorithm

The algorithm is finite due to the lexicographically decreasing nature of the “constant value” row. At each iteration the pivot column is scaled by $\lfloor \frac{\bar{a}_{s,0}}{\lambda} \rfloor$, which is less than zero. Because each column (besides the “constant value” column) is lexicographically greater than zero (since a larger column is never subtracted from it—hence the derivation of u_j), the “constant value” column will decrease lexicographically after each iteration.

Thus, the only way for the algorithm to loop forever would be for there to be an infinite decreasing sequence of “constant value” columns. Since a feasible solution exists, $\bar{a}_{0,0}$ is bounded below by the total cost of this solution. Thus, for the algorithm to loop, $\bar{a}_{i,0}, i > 0$ must decrease forever. However, one can show that each element must have a lower bound.

Examine the element $\bar{a}_{1,0}$. If it goes below zero, the first row is eligible as the source row. When this row is chosen as the source row, its value increases; to ensure

the column as a whole lexicographically decreases, then $\bar{a}_{0,0}$ must decrease. If $\bar{a}_{1,0}$ decreased forever, column one would be the source row an infinite number of times, and $\bar{a}_{0,0}$ would decrease forever as well. But $\bar{a}_{0,0}$ is known to be bounded below, therefore $\bar{a}_{1,0}$ must be bounded below too. Repeat this argument for all $\bar{a}_{i,0}, i > 1$ to prove finiteness of the algorithm.

3.8.2 Source Row Selection

The above proof does rely on the assumption that if a row is negative it will eventually be chosen. The source row selection rule presented in Section 3.6.2 does not abide by this rule.⁷ Using the “most negative” rule will result in cycling in a significant number of large all-integer simplex problems. One potential rule would be to choose the first row with a negative entry in the “constant value” column. Unfortunately, while this change solves the problem of cycling, it increases the number of iterations to an intolerable amount.

Upon finding the same inefficiency, Gomory [12] introduced a “look ahead” rule to determine the source row. The objective of this rule to choose a row that will result in the largest pivot column (lexicographically speaking), thus driving down the “constant value” column as far as possible. There are three steps to the rule:

1. List the columns (as usual, ignoring the “constant value” column) in descending lexicographical order. Assign each column a rank $(1, 2, \dots, n)$ based on that ordering. Let this rank be $C(j)$.
2. For every row with $\bar{a}_{i,0} < 0$, assign the row a rank $R(i) = \text{maximum } C(j)$ where $j \in \{j | \bar{a}_{i,j} < 0\}$.
3. Select source row $s = \text{argmin}_i R(i)$.

I break ties by choosing the smallest i (i.e., the first row).

⁷Indeed, as Solow [25] points out, in LP simplex the “most negative” rule, while lowering the number of iterations needed, is vulnerable to cycling as well. However, he also asserts that cycling rarely happens; a claim my experiential evidence supports. Bound and enumeration (under this selection regime) never failed to solve the LP relaxation problems.

While the Gomory rule does greatly speed up the algorithm, cycling does occur because rows are not guaranteed to be chosen. To solve this dilemma between efficiency and consistency, I introduce non-determinism into the program. For two-thirds of the iterations I use the Gomory criterion and one-third of the time I choose the first eligible row. The two-thirds/one-third split is largely arbitrary, only a small amount of searching was done in an attempt to find the optimal ratio. Regardless of the optimal percentage, this configuration never leads to cycling and solves weighted voting games in a reasonable amount of time (see Sections 4.5 and 4.6).

3.8.3 Cut Formation and Deletion

Sources conflict [22] [10] on whether to delete a newly formed cut row after it has been used to pivot. Even Firla, who recommends keeping the rows in the tableau notes that they should not be used as source rows. Thus, they are useful only for record keeping; in large problems, such as 14-party parliaments, it is wise to delete the rows after pivoting. Therefore, the space needed for the tableau does not grow and remains at $O(n^2 \cdot m)$ for the duration of the algorithm.

The same sources do agree, however, for the case when $\lambda = 1$. In this situation, the added cut would be the same as the source row; thus, the source row can be used as the pivot row and there is no need to create the additional cut.

3.8.4 All-Integer Simplex Example

Example Solve the weighted voting game [46;32,24,18,17].

1. *Coalition Enumeration.* There are four minimal winning coalitions: $\{32,24\}$, $\{32,18\}$, $\{32,17\}$, $\{24,18,17\}$. There are no tying coalitions.
2. *Ranking.* Labelling the parties in descending order yields $rank(1) = 1$, $rank(2) = 2$, $rank(3) = 2$, and $rank(4) = 2$.

3. Constraints.

$$\begin{aligned}
 -w_1 - w_2 + w_3 + w_4 &\leq -1 \\
 -w_1 + w_2 - w_3 + w_4 &\leq -1 \\
 -w_1 + w_2 + w_3 - w_4 &\leq -1 \\
 w_1 - w_2 - w_3 - w_4 &\leq -1 \\
 -w_1 + w_2 &\leq -1 \\
 -w_2 + w_3 &\leq -1 \\
 -w_3 + w_4 &\leq -1 \\
 -w_4 &\leq -1
 \end{aligned}$$

4. All-Integer simplex

Source rows are marked with a \rightarrow , pivot elements are in **bold**, and the solution in the optimal tableau is *italicized*. Note the monotonically decreasing negative total cost in $\bar{a}_{0,0}$ and the lexicographically decreasing “constant” column.

#1	1	$(-x_1)$	$(-x_2)$	$(-x_3)$	$(-x_4)$
$(-z)$	0	1	1	1	1
x_1	0	-1	0	0	0
x_2	0	0	-1	0	0
x_3	0	0	0	-1	0
x_4	0	0	0	0	-1
x_5	-1	-1	-1	1	1
x_6	-1	-1	1	-1	1
$\rightarrow x_7$	-1	-1	1	1	-1
x_8	-1	1	-1	-1	-1
x_9	-1	-1	1	0	0
x_{10}	-1	0	-1	0	0
x_{11}	-1	0	0	-1	0
x_{12}	-1	0	0	0	-1

Source row criterion: Gomory; potential rows: $\{7,9\}$; source row, $s = 7$.

Potential pivot columns: $\{1,4\}$; $u_1 = 1$, $u_4 = \infty$; $\lambda = \lambda_1 = 1$, $\lambda_4 = 0$; pivot col., $p = 1$
 $\bar{a}_{r,p} = -1$, so there is no need to add a cut.

#2	1	$(-x_7)$	$(-x_2)$	$(-x_3)$	$(-x_4)$
$(-z)$	-1	1	2	2	0
x_1	1	-1	-1	-1	1
x_2	0	0	-1	0	0
x_3	0	0	0	-1	0
x_4	0	0	0	0	-1
x_5	0	-1	-2	0	2
x_6	0	-1	0	-2	2
x_7	0	-1	0	0	0
$\rightarrow x_8$	-2	1	0	0	-2
x_9	0	-1	0	-1	1
x_{10}	-1	0	0	0	0
x_{11}	-1	0	0	-1	0
x_{12}	-1	0	0	0	-1
x_{13}	-1	0	0	0	-1

Source row criterion: any negative; potential rows: $\{8,10,11,12\}$; source row, $s = 8$.
Potential pivot column: $\{4\}$; $u_4 = 1$; $\lambda = \lambda_4 = 2$; pivot col., $p = 4$

#3	1	$(-x_7)$	$(-x_2)$	$(-x_3)$	$(-x_{13})$
$(-z)$	-1	1	2	2	0
x_1	0	-1	-1	-1	1
x_2	0	0	-1	0	0
x_3	0	0	0	-1	0
x_4	1	0	0	0	-1
x_5	-2	-1	-2	0	2
$\rightarrow x_6$	-2	-1	0	-2	2
x_7	0	-1	0	0	0
x_8	0	1	0	0	-2
x_9	-1	-1	0	-1	1
x_{10}	-1	0	-1	0	0
x_{11}	-1	0	0	-1	0
x_{12}	0	0	0	0	-1

Source row criterion: Gomory; potential rows: $\{6,9\}$; source row, $s = 6$.
Potential pivot columns: $\{1,3\}$; $u_1 = 1, u_3 = 3$; $\lambda = \lambda_1 = 1, \lambda_3 = \frac{2}{3}$; pivot col., $p = 1$
 $\bar{a}_{r,p} = -1$, so there is no need to add a cut.

#4	1	$(-x_6)$	$(-x_2)$	$(-x_3)$	$(-x_{13})$
$(-z)$	-3	1	2	0	2
x_1	2	-1	-1	1	-1
x_2	0	0	-1	0	0
x_3	0	0	0	-1	0
x_4	1	0	0	0	-1
x_5	0	-1	-2	2	0
x_6	0	-1	0	0	0
x_7	2	-1	0	2	-2
x_8	-2	1	0	-2	0
$\rightarrow x_9$	1	-1	0	1	-1
x_{10}	-1	0	-1	0	0
x_{11}	-1	0	0	-1	0
x_{12}	0	0	0	0	-1

Source row criterion: Gomory; potential row: $\{9\}$; source row, $s = 9$.

Potential pivot columns: $\{1,4\}$; $u_1 = 1, u_4 = 3$; $\lambda = \lambda_1 = 1, \lambda_4 = \frac{1}{3}$; pivot col., $p = 1$
 $\bar{a}_{r,p} = -1$, so there is no need to add a cut.

#5	1	$(-x_9)$	$(-x_2)$	$(-x_3)$	$(-x_{13})$
$(-z)$	-5	1	2	0	2
x_1	3	-1	-1	1	-1
x_2	1	0	-1	0	0
x_3	0	0	0	-1	0
x_4	1	0	0	0	-1
x_5	2	-1	-2	2	0
x_6	-2	-1	0	0	0
x_7	1	-1	0	2	-2
$\rightarrow x_8$	0	1	0	-2	0
x_9	-1	-1	0	1	-1
x_{10}	0	0	-1	0	0
x_{11}	-1	0	0	-1	0
x_{12}	0	0	0	0	-1
x_{14}	-1	0	0	-1	0

Source row criterion: Gomory; potential row: $\{8,11\}$; source row, $s = 8$.

Potential pivot column: $\{3\}$; $u_3 = 1$; $\lambda = \lambda_3 = 2$; pivot col., $p = 3$

#6	1	$(-x_9)$	$(-x_2)$	$(-x_{14})$	$(-x_{13})$
$(-z)$	-5	1	2	0	2
x_1	2	-1	-1	1	-1
x_2	1	0	-1	0	0
x_3	1	0	0	-1	0
x_4	1	0	0	0	-1
x_5	0	-1	-2	2	0
x_6	0	-1	0	0	0
x_7	0	-1	0	2	-2
x_8	0	1	0	-2	0
x_9	0	-1	0	1	-1
x_{10}	0	0	-1	0	0
x_{11}	0	0	0	-1	0
x_{12}	0	0	0	0	-1

Optimal tableau:

$$w_i = \bar{a}_{0,j} \quad (1 \leq j \leq 4) = [2, 1, 1, 1]$$

3.9 Worst Case Orders of Growth of Algorithms

Despite the development of polynomial-time algorithms to solve linear programming problems [23], the simplex algorithm, which grows exponentially in the worst case, remains quite a popular algorithm [25]. In the worst case, where the variables and constraints conspire to form a space with 2^{m+n} vertices, the vertices can be ordered so that simplex moves linearly from one to the next. (Here, m is the number of equations, and n is the number of variables.) Thus, worst case, simplex takes $O(2^{m+n})$ [28]. However, problems take only $O(m+n)$ iterations on average [23].

In the specific case of majoritarian voting games, the number of variables, n , and the number of equations, m , are closely related. The number of equations is the number of minimal winning coalitions, plus the number of tying coalitions, plus the number of non-dummy parties (for the rank constraints). Upper bounds for those variables are 2^{n-1} , $\frac{2^n}{\sqrt{n}}$, and n respectively.⁸ How the number of minimal winning and

⁸Calculating the upper bound of $(2^n/\sqrt{n})$ for tying coalitions is involved. Given n parties with $t = q - 1 = \frac{1}{2} \sum s_i$ and $h = \frac{n}{2}$, the largest number of combinations of parties that add up to t is when $\forall i. s_i = c$ and $t \bmod c = 0$, where c is a constant integer. In this case, the number of tying coalitions is $\binom{n}{h} = n!/(h!^2)$. Taking the \ln : $\ln(n!) - 2\ln(h!)$. Using Stirling's approximation

tying coalitions vary with the number of parties on average is discussed in Section 4.2. Also, note that the use of the Beale tableau adds n rows to the matrix, which would normally only consist of the m equation rows. Let the total number of simplex rows (or constraints) be c ; thus, $c=O(m+n)$, which is also the order of growth of the average simplex execution.

Generalized integer programming is an NP-complete problem, and thus a polynomial time algorithm for the minimum integer weight search problem is unlikely to be found [23]. To calculate the order of growth in time for bound and enumerate, the number of iterations and the number of combinations per iteration must be taken into account. Given n parties with r ranks, the minimum total cost that the original root node could return is $(n-r) + \sum r$. The $\sum r$ term is derived from the fact that the difference between adjacent rank values is bounded below by one. The $(n-r)$ term signifies the additional parties that belong to non-exclusive ranks; the minimum weight for these parties is one. The maximum total cost is $2q$ (i.e., the total number of seats), thus the order of growth of the maximum number of iterations is $O(q-r^2)$. At each iteration, the number of sequences enumerated is limited by the number of monotonic combinations of weights that add up to k . This is somewhat analogous to the maximum number of tying coalitions. Indeed, as k approaches q the number of combinations that sums to k approaches half⁹ that of the potential number of tying coalitions. Thus, the upper bound for combinations is $\frac{2^{n-1}}{\sqrt{n}}$. The worst case order of growth for bound and enumerate is $O((q-r^2) * \frac{2^{n-1}}{\sqrt{n}})$ —clearly exponential.

The order of growth of all-integer simplex directly depends on the number of cuts needed before an optimal solution is found. Schrijver [23] proved that the number of cuts needed can not be bounded by a polynomial number. That is not to say that on average all-integer simplex cannot be exponential, but that, worst case, integer simplex will be exponential. Cook, Coullard, and Turan [8] then showed that, for empty spaces, the number of cuts is bounded by $O(c^{3c})$. (That exponential result

(let $k = \frac{1}{2} \ln(2\pi)$): $\approx n \ln(n) + \frac{1}{2} \ln(n) - n + k - 2h \ln(h) - \ln(h) + 2h - 2k = n \ln(n) + \frac{1}{2} \ln(n) - n \ln(h) - \ln(h) - k = \ln(n^{n+(1/2)}/h^{n+1}) - k = \ln(2^{n+1}/\sqrt{n}) - k = \ln(2^{n+1}/\sqrt{2\pi n})$. Undoing the \ln : $2^{n+1}/\sqrt{2\pi n} = (\sqrt{2}/\sqrt{\pi}) * (2^n/\sqrt{n}) < (2^n/\sqrt{n})$. For odd n , use $(n-1)$ to keep h as an integer.

⁹Half since tying coalitions are unique in that their complement also meets the required sum. This is not true for any other k , thus cutting the pool of potential sequences in half.

implies that an all-integer simplex process at each bound and enumerate iteration would not be helpful for determining emptiness of fixed-cost spaces). Further research by Cook, Gerards, Schrijver, and Tardos [9] showed that for any space, the number of cuts needed only depends on c . Thus, a reasonable assumption would be that the order of growth of simplex, in the worst case, would be $O(c^c)$. In the next chapter, the average number of cuts and time needed for a solution is analyzed.

Chapter 4

Results and Analysis

4.1 Methodology

By using these algorithms for finding the minimum integer weights for arbitrary games, I now examine (1) different properties of games, (2) the proportionality of minimum integer weights, (3) the relative strengths and weaknesses of the algorithms. The following analysis is based on randomly-generated games. For empirical results, see Ansolabehere [1] and Strauss [26].

Random games were generated based on two parameters: the number of parties and the maximum number of seats. The number of parties (denoted n) ranges from three to 13; the maximum seat size (denoted t) has five discrete values: 5, 20, 50, 100, and 250. For each party and maximum seat size, 500 random games were generated. Thus, games are laid out on two axes; I refer to games with relatively many parties as “large games” and games with a large maximum seat parameter as “many-seat games.” (The analogous terminology for smaller games is “small games” and “few-seat games.”) In all cases, games were solvable by at least one algorithm; and, games that could be solved by only one algorithm composed a small minority.

When one party’s weight alone is at least a majority of the total weight, that player is considered a “dictator” and the game is a “dictator game.” A probabilistic analysis indicates that for three players with randomly chosen seats (and no maximum), the

probability of the resulting game including a dictator is 50%.¹ Indeed, almost half of the large-seat, three-player stochastic games include dictators. The data also confirm the probabilistic facts that as the number of players increases, the rarer dictator games become (Figure 4-1); and, that as the maximum number of seats increases, the more prevalent dictator games become (Figure 4-2). Since game theorists are interested only in games for which multi-player coalitions are formed, dictator games are excluded from further analyses. This proscription has a significant effect only on small player games.

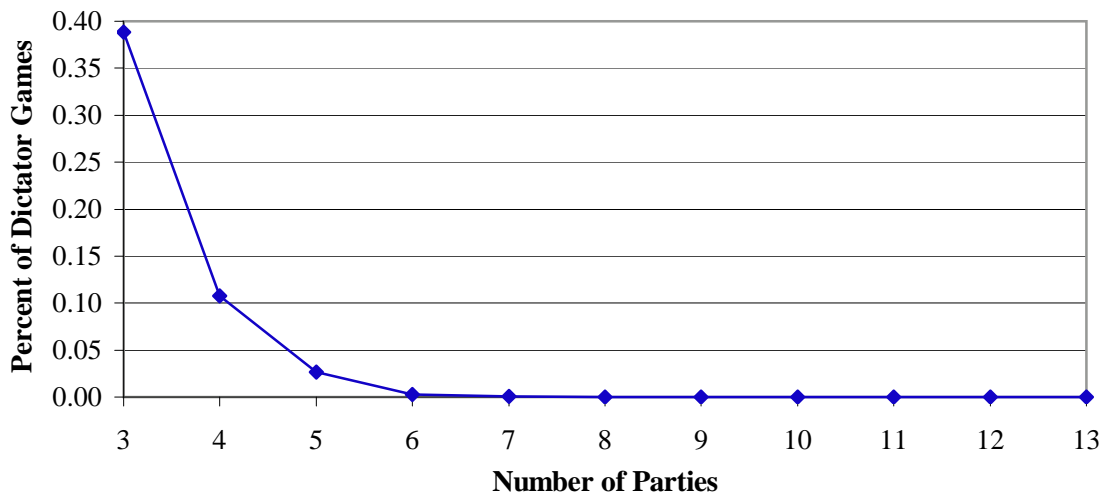


Figure 4-1: Incidence of Dictator Games by Number of Parties

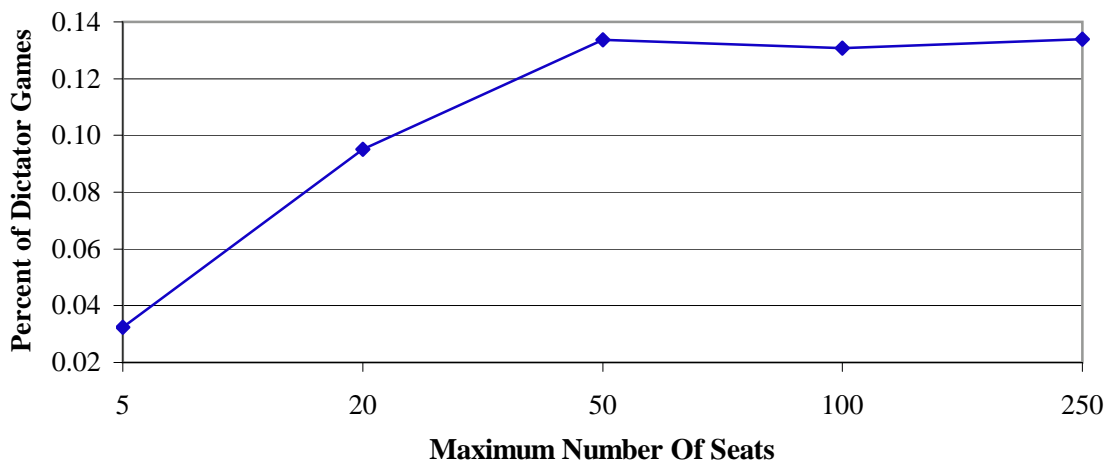


Figure 4-2: Incidence of Dictator Games by Maximum Seats ($n \leq 7$)

¹To calculate this, find $\Pr(Z - W > 0) + \Pr(Z - W' > 0)$ where $W = X + Y$, $W' = |X - Y|$, and X, Y , and Z are random variables uniformly distributed over $[0, 1]$.

4.2 Number of Coalitions and Constraints

The reason that larger games are harder to solve (see Section 4.5) is because more parties lead to more coalitions. Since the running time of all-integer simplex is greatly affected by the number of coalitions, how quickly the number of coalitions grows needs to be examined. Figure 4-3 shows that the relationship between coalitions and parties is exponential. Running ordinary least squares on the log of the number of coalitions indicates that the number of coalitions grows by approximately $O(1.7^n)$ (Table 4.1).

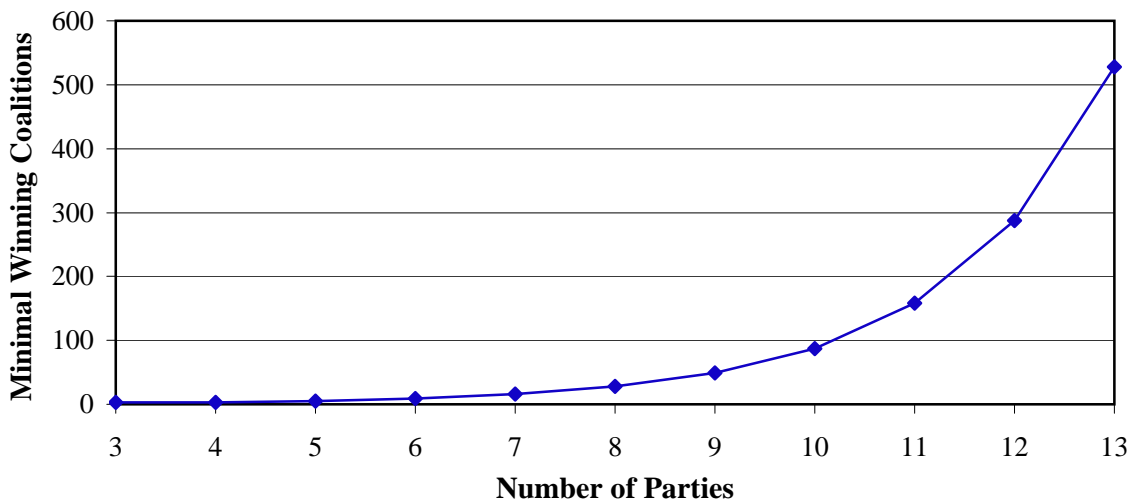


Figure 4-3: Average Number of Min. Win. Coalitions by Number of Parties

Interestingly, the number of coalitions decreases as the maximum seats allowed per party increases. The reason is that the sparseness of the seat values in many-seat games allows a few parties to dominate, thus producing fewer coalitions. A linear relationship fits the data slightly better than an exponential relationship, though more in-depth study would be needed to verify this claim. Another notable property is that the number of *unique* coalitions increases as party size rises. In this case, the dense seat-space in few-seat games produces many fewer ranks than parties, thus decreasing the number of unique coalitions and outweighing the opposing trend shown in Figure 4-4.

A more probing analysis of coalitions versus maximum seat size only produces hazier results. For instance, if one only looks at games with specific parameters (e.g.,

$n = 11$ and $t = 100$), then the number of coalitions *increases* over the average number of seats per party. This claim appears to be in direct contradiction with the finding in the preceding paragraph. However, as one increases the maximum seat parameter, the positive relationship between coalitions and average seat size becomes more “bottom heavy,” bringing the group’s average down. To detail this local relationship a bit more, as maximum seat size increases the slope of this positive relationship decreases, while increasing n has an opposite effect. Analysis shows that the most likely reason for this relationship is that the more relatively small parties are in a game, the fewer coalitions there will be, since coalitions will need more parties in them to reach the quota.

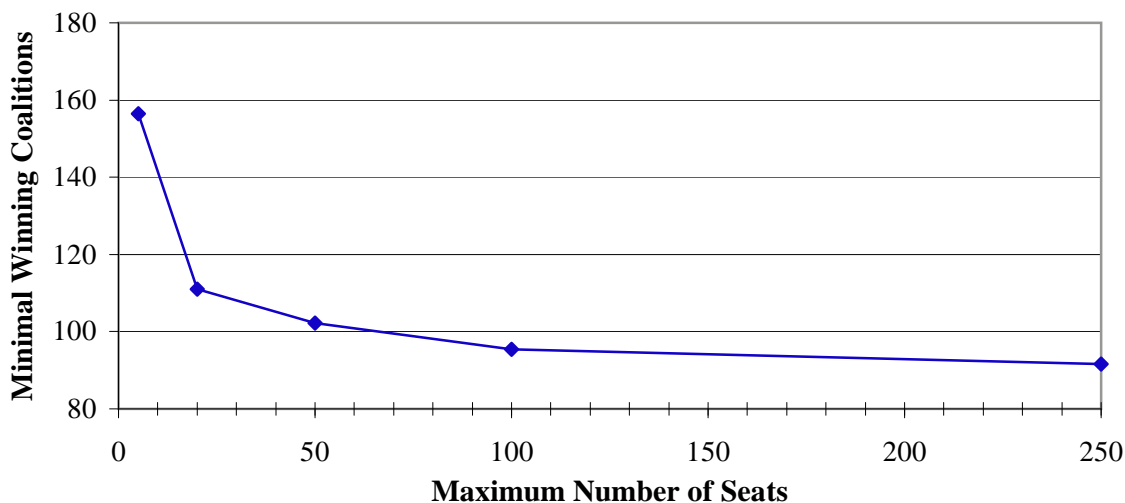


Figure 4-4: Average Number of Min. Win. Coalitions by Maximum Seats

The number of tying coalitions is also affected by the density of the seat-space in games. As the discussion in Section 3.9, the scenario which results in the most tying coalitions is when all the parties have equal weight. As party size spreads out over a larger potential space (i.e., when the maximum seat parameter increases), the number of tying coalitions decreases. This principle is illustrated in Figure 4-5.

The number of coalitions in a game affects both the subproblem of coalition enumeration and of solution searching. Since the number of minimal winning coalitions grows exponentially on the size of n , any algorithm—no matter how clever—that enumerates all the minimal winning coalitions will grow exponentially. There are a

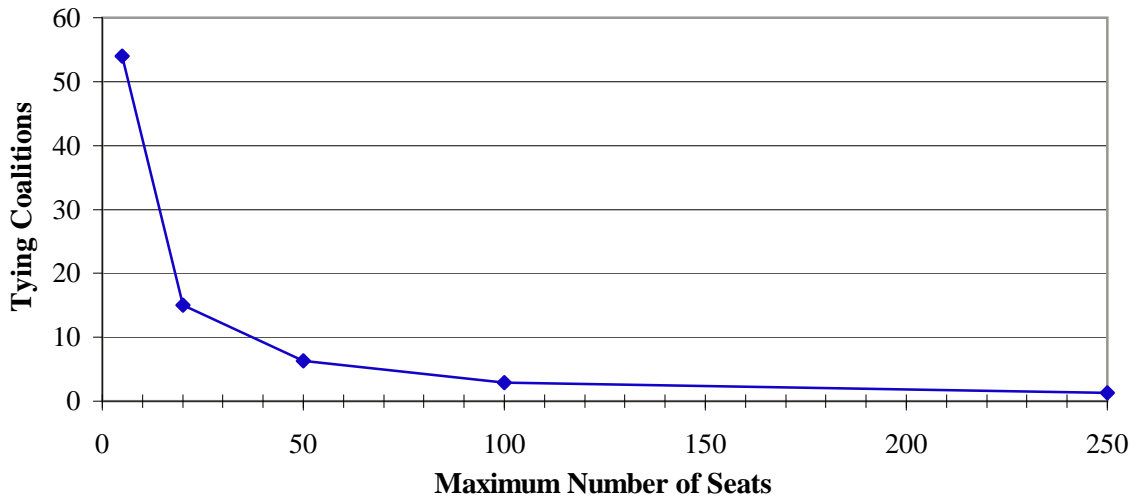


Figure 4-5: Average Number of Tying Coalitions by Maximum Seats

few potential ways to avoid this problem. One, even minimal winning coalitions can be redundant. For instance, let a minimal winning coalition S have as its smallest two seat values s_i and s_j respectively. For all players p_k , where $s_i \geq s_k > s_j$, $S' = S - \{p_j\} + \{p_k\}$ will be a minimal winning coalition as well. Each coalition of type S coalition implies all S' minimal winning coalitions; thus, S' coalitions need not be enumerated. While the number of defining coalitions would probably still be exponential on average, further such insights might prove very fruitful in reducing the time needed for coalition enumeration.

Another potential method for enumerating fewer coalitions would be a randomized, bounded error, algorithm. Perhaps some polynomial number of coalitions could be picked and checked to see if they were minimal winning. Clever algorithms could probabilistically (or even deterministically) avoid picking coalitions that were obviously not minimal winning. Sometimes a crucial coalition would be missed, thus producing weights that defined a different game. A difficult task would be knowing whether one had indeed stumbled upon the correct weights or whether the algorithm should be run again.

For the second subproblem, specifically when a simplex technique is employed, the relevant question is how the number of constraints grow over the size of games. To find the number of rows present in the Beale Tableau, one sums the number of mini-

mal winning coalitions, tying coalitions, and double the non-dummy parties (needed for both the identity sub-matrix at the top of the tableau and ranks constraints). As Figures 4-6 illustrates, the number of rows is dominated by the number of minimal winning coalitions, and thus grows exponentially on the number of parties and decreases polynomially on the maximum seat parameter. Overall, the number of rows

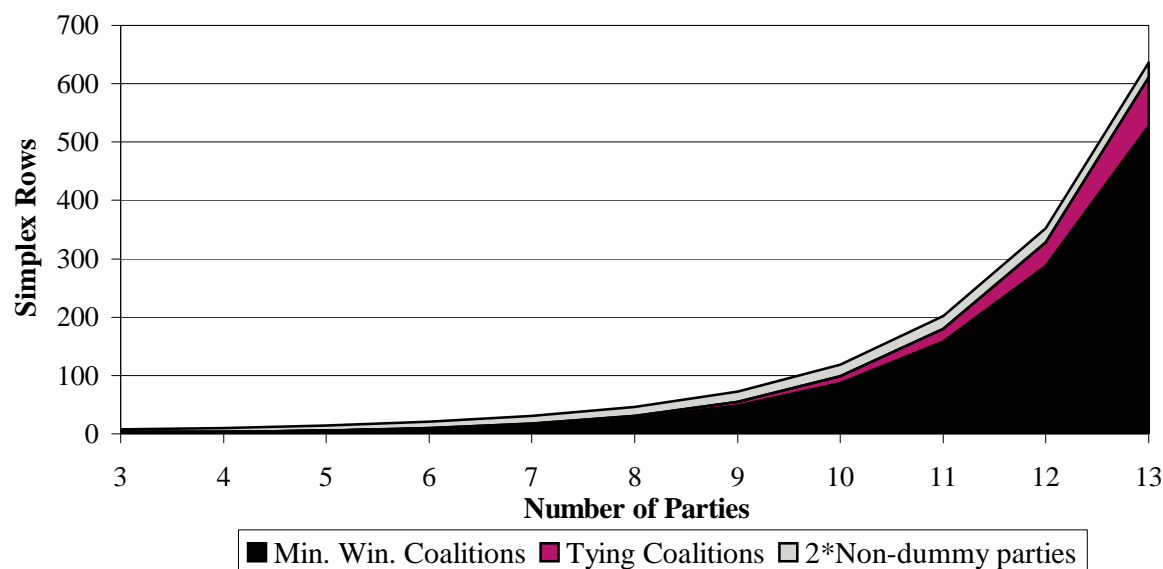


Figure 4-6: Composition of Simplex Rows Averaged over Number of Parties

Dependent Variable	Dep. Var. = $\alpha\beta^n t^\gamma$			
	α	β	γ^*	R^2
Min. Win. Coalitions	0.366	1.721	-	0.945
Min. Win. Coalitions	0.544	1.723	-0.110	0.952
Simplex Rows	1.721	1.533	-	0.925
Simplex Rows	2.776	1.535	-0.132	0.943

All coefficients significant at 99% confidence level

*Assumed zero when excluded

Table 4.1: OLS Regressions for Minimal Winning Coalitions and Simplex Rows

4.3 Proportionality of Minimum Integer Weights

A major impetus for this research was to discover whether minimum integer weights, and thus potentially power, follow Gamson's Law of proportionality. Empirical ev-

idence indicated that for games with few parties, larger parties would have disproportionately less power, while for games with many parties, power was proportional. While prior research [26] confirmed this result with incomplete data, given the importance of this question, it is worth re-examining.

The data, this time complete and extended, once again supports Gamson’s Law for large parties (see Figure 4-7). In smaller games, there is a “relative weakness effect”: the coefficient on seat share is below one, and the constant is positive. But as the number of parties increases, these values approach unity and zero. The maximum number of seats has little impact on these numbers (with the exception of small, few-seat games)

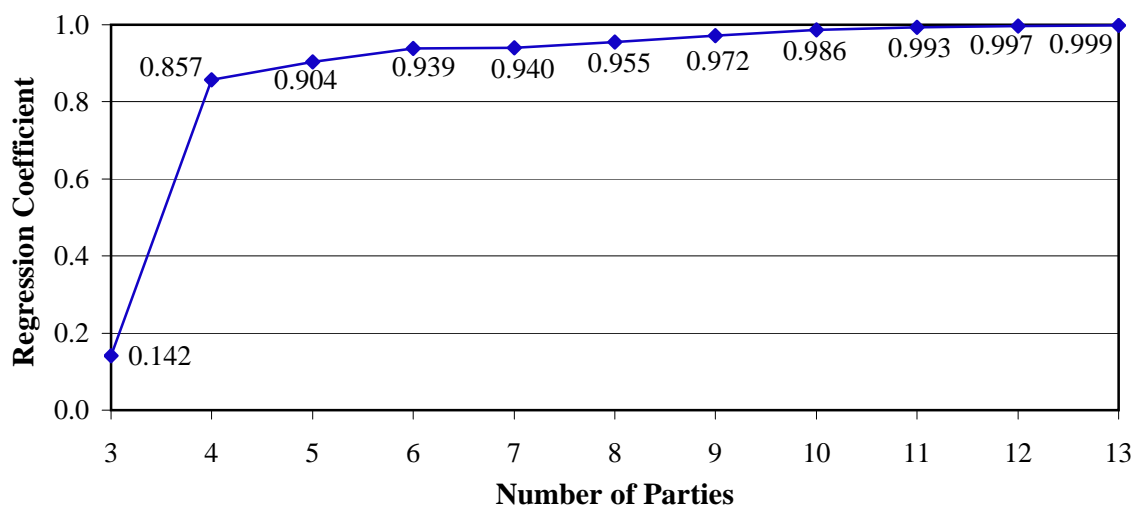


Figure 4-7: Weight Share Regression Coefficients

An appropriate question to raise is whether the almost perfect correlation between weight share and seat share ($R^2 = 0.9997$ when $n = 13$) is due to the congruence of minimum integer weights and seats. Indeed, as the number of parties increase, the less often seat values need to be reduced to find the minimum integer weights. With 13 parties, 68 percent of games had this property. However, even after removing those games *and* removing those games with similar weights and seats², the change in the coefficient values presented in Figure 4-7 was less than one percent³ for all $n > 3$.

²Let the percent of weight reduction be $(\text{total seats} - \text{total weights}) / (\text{total seats}) = (\sum s_i - \sum w_i) / (\sum s_i) = \delta$. Consider games with $\delta < .10$ to have “similar” weights and seats.

³Let percent change in coefficient values be $(\text{old coefficient} - \text{new coefficient}) / (\text{old coefficient})$.

4.4 Homogeneity of Games

From Von Nuemann [29] to Isabel [15] in the early years of voting game research to Morelli [18] and Snyder [27] more recently, scholars have had a penchant for examining properties of homogeneous games. One reason is that the mathematics behind homogeneous games is easier, thus leading to the second reason: there are more interesting properties to discover. As noted in previous work [26], the prevalence of homogeneous games decreases as games become larger. Now with complete data, the view (Figure 4-8) is sharper

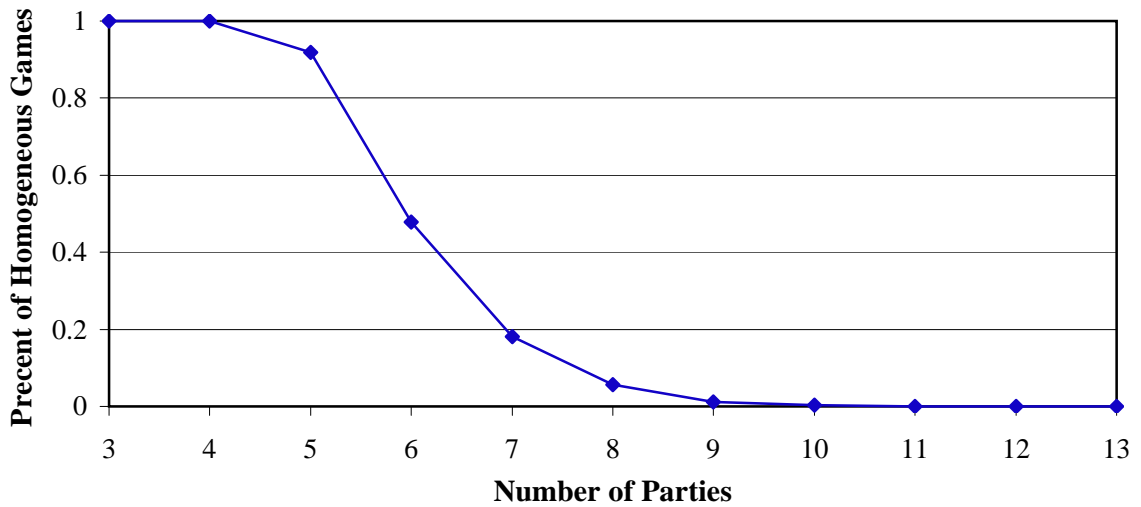


Figure 4-8: Percent of Homogeneous Games by Number of Seats

A simple probit analysis reveals a coefficient of 0.976 on n and 1.94×10^{-3} on t (Std. Err.'s: 1.27×10^{-2} , 1.69×10^{-4} ; pseudo- $R^2=0.741$). Taking the percent of homogeneous cases at each party-seat value and running a Gompertz nonlinear regression on n yields:

$$\text{Percent Homogeneous} = .0143 + 1.047 * e^{-e^{1.123(n-6.342)}} \quad R^2 = 0.979 \quad (4.1)$$

$$\text{Standard errors:} \quad (0.0117) \quad (0.0340) \quad (0.120) \quad (0.0665)$$

While there might be some theoretical insight to be gained for why a Gompertz “s-curve” fits the data so well, the important message the data provide is that the number of homogeneous games approaches zero in large games. Thus, researches must

refocus their efforts to nonhomogeneous games if their analyses are to be generally useful.

4.5 Running Times of Algorithms

Both all-integer simplex and bound and enumerate algorithms were clearly faster than the branch and bound method used in prior research [26]. While simplex is faster than bound and enumerate on average, there are some games for which bound and enumerate will find the solution but all-integer simplex will choke on. These games often have the attribute that their initial seats are the minimum integer weights for the game.⁴ In these cases, bound and enumerate finds the solution when it examines the root node while all-integer simplex starts at the (infeasible) origin and works its way toward the solution. In these cases, bound and enumerate is much faster.

Given these different approaches to solve the same problem, a third algorithm, “augmented integer simplex,” was developed. This new method first runs the root LP relaxation problem, as in bound and enumerate. Only if an integer solution is not found, then all-integer simplex is executed with the additional constraint that the total cost must be greater than the ceiling of the root node’s cost.

To compare the three methods, 50 random games were generated for each combination of the two parameters of number of parties and maximum seats per party. To get a better sense of how game difficulty changed on the second parameter, more maximum seat levels were added. The algorithms were given ten minutes (excluding coalition enumeration) to solve each game. For many-seat, 12-player games bound and enumerate struggled, solving only 60 percent of games attempted. Thus, larger games were not attempted for that algorithm. Similarly, all-integer simplex, started significantly slowing on 13-player games, and the algorithm was discontinued after that level. But, augmented integer simplex was still going strong with 14-player games—solving over 90 percent of games in under ten minutes and needing (on average) fewer than *nine seconds* per solved game (Figure 4.2).

⁴Or, more generally, when the seats are the minimum weights given the extra rank constraints.

n	t : Alg.*	5			50			100		
		Attribs [‡]	Time ^b	S% [‡]	Attribs	Time	S%	Attribs	Time	S%
8	ISim	42	37	All	42	21	All	53	25	All
	BaE	1; 0	55	All	8; 2,498	406	All	6; 1,299	235	All
	AIS	13; 82%	18	All	33; 62%	22	All	22; 80%	14	All
9	ISim	54	78	All	68	60	All	79	66	All
	BaE	0; 0	56	All	14; 41,214	1,852	All	10; 9,910	1,027	All
	AIS	3; 94%	14	All	51; 56%	55	All	37; 74%	39	All
10	ISim	75	266	All	121	208	All	182	324	All
	BaE	0; 0	220	All	21; 3,315	6,230	All	33; 86,953	11,376	All
	AIS	4; 96%	35	All	79; 58%	157	All	91; 56%	187	All
11	ISim	62	381	All	329	1,110	All	617	2,621	All
	BaE	0; 0	237	All	27; 5,272	23,595	All	65; 84,347	55,470	98%
	AIS	1; 98%	50	All	182; 54%	898	All	269; 50%	1,387	All
12	ISim	71	1,013	All	1030	8,682	All	1300	13,167	98%
	BaE	0; 0	127	All	27; 391	61,265	98%	45; 30,463	67,276	88%
	AIS	0; All	128	All	126; 62%	1,254	All	246; 60%	2,666	All
13	ISim	95	2,891	All	1248	20,303	All	1850	29,400	80%
	BaE	-	-	-	-	-	-	-	-	-
	AIS	0; All	300	All	247; 72%	4,716	All	245; 57%	3,964	98%
14	ISim	-	-	-	-	-	-	-	-	-
	BaE	-	-	-	-	-	-	-	-	-
	AIS	0; All	767	All	20; 96%	1,862	All	194; 72%	6,810	94%

n	t : Alg.	150			200			250		
		Attribs	Time	S%	Attribs	Time	S%	Attribs	Time	S%
8	ISim	47	27	All	44	21	All	39	19	All
	BaE	5; 364	225	All	6; 4,322	273	All	2; 1,073	78	All
	AIS	16; 82%	12	All	15; 82%	10	All	7; 94%	4	All
9	ISim	109	101	All	102	98	All	86	79	All
	BaE	14; 111,290	3,447	All	14; 107,020	3,476	All	14; 422,133	11,971	All
	AIS	40; 74%	38	All	32; 74%	37	All	26; 82%	24	All
10	ISim	279	508	All	206	318	All	191	322	All
	BaE	44; 232,567	17,962	All	27; 208,941	11,690	90%	27; 504,574	21,098	96%
	AIS	84; 54%	165	All	71; 66%	141	All	63; 70%	126	All
11	ISim	804	2,692	All	616	2,956	All	710	2,668	All
	BaE	65; 393,944	72,060	94%	71; 751,520	64,802	94%	40; 790,236	54,581	88%
	AIS	127; 52%	519	All	138; 54%	459	All	87; 66%	322	All
12	ISim	1483	11,830	All	1759	10,806	90%	1266	10,962	94%
	BaE	67; 418,095	129,349	74%	48; 62,867	69,633	66%	15; 17,722	20,946	60%
	AIS	403; 48%	4,282	96%	198; 54%	1,694	All	1,372; 57%	11,349	98%
13	ISim	1384	30,378	82%	1500	24,358	80%	1899	33,147	80%
	BaE	-	-	-	-	-	-	-	-	-
	AIS	611; 48%	15,874	All	145; 68%	2,300	94%	176; 63%	2,831	92%
14	ISim	-	-	-	-	-	-	-	-	-
	BaE	-	-	-	-	-	-	-	-	-
	AIS	643; 63%	23,165	98%	496; 54%	14,257	96%	268; 61%	8,679	92%

*All Integer Simplex, Bound and Enumerate, Augemnet Integer Simplex.

[‡]ISim : Number of cuts added. BaE : Number of Iterations; Number of enumerations.

AIS : Number of iterations; Percent solved by root node.

^bTime is in milliseconds. [‡]Percent of games solved.

Table 4.2: Time Trials for the Three Algorithms

A simpler demonstration and further discussion of the dominance of augmented simplex is illustrated in Section 4.7.

4.6 Average Iterations for All-Integer Simplex

Given that, in the worst case, all-integer simplex grows exponentially with the number constraints (c) [9] [8], and that, on average, non-integer (normal) simplex scales linearly [23], an intriguing question is how all-integer simplex grows on average. The answer is exponentially, but barely. I ran 2000 stochastic games with between eight and 12 parties (with a variable maximum seat parameter). Then I took a moving average based on the number of simplex rows, not on the number of data points, to avoid biasing the results to the denser smaller games. The result is show in Figure 4-9. The number of iterations all-integer simplex requires is on the order of $O(1.01^c)$, which is about as slow as a function can grow and still be exponential. However, the data are certainly not linear. No matter how the data were adjusted, be it by removing outliers or changing the moving average, an exponential model always fit better than a linear one. This finding is unfortunate news for the all-integer simplex algorithm, since, as noted in Section 4.2, the constraints already scale exponentially on the number of players

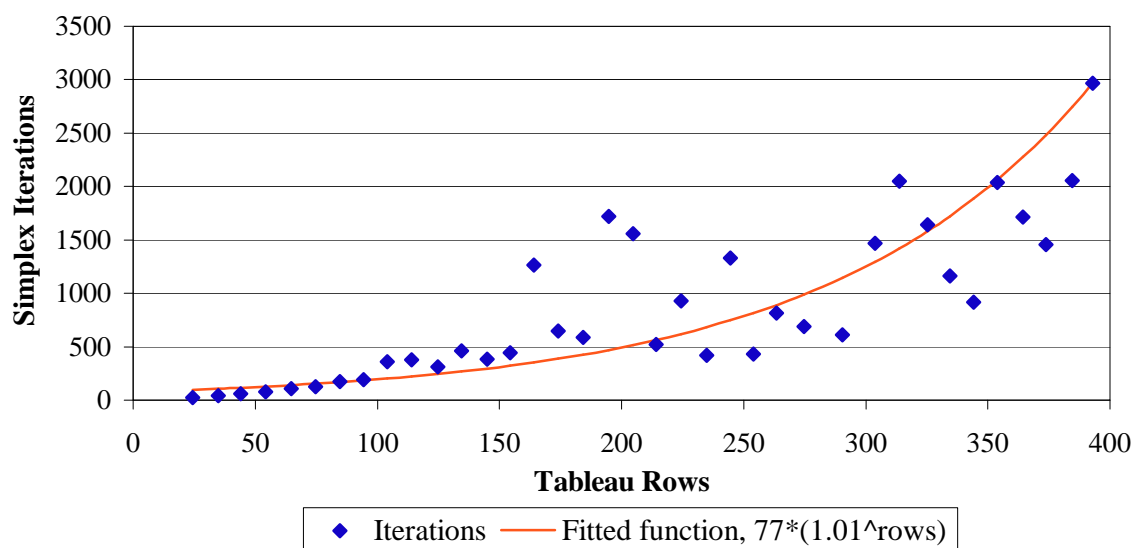


Figure 4-9: Exponential Growth of All-Integer Simplex on Number of Constraints

4.7 Solving an Arbitrary Game

If all-integer simplex scales poorly with large games, then augmented integer simplex will similarly slow on games that are not solved by the root node. As games get larger, however, the probability increases that the *a priori* seat values are the minimum weights (with the additional rank constraints). This property enables the root node to solve more games. In this tug-of-war between the slow simplex process and the quick root node, thankfully, the root node is the winner. Eager to test augmented integer simplex's limits, I extended the analysis to 17 games (Figure 4-10). Note that the percent of games solved by augmented integer simplex actually *increases* between 14 and 17 parties.⁵ Given these results, finding the minimum integer weights for *any* game is clearly only a matter of resources and patience. For very large games the bottleneck becomes coalition enumeration. While, coalition enumeration is exponential, there is neither uncertainty of whether it will terminate nor of how long it will take to terminate. Since, the world's fastest computers execute over 10^{12} instructions per second, even the Electoral College, with (worst case) 10^{15} winning coalitions, seems well within reach.

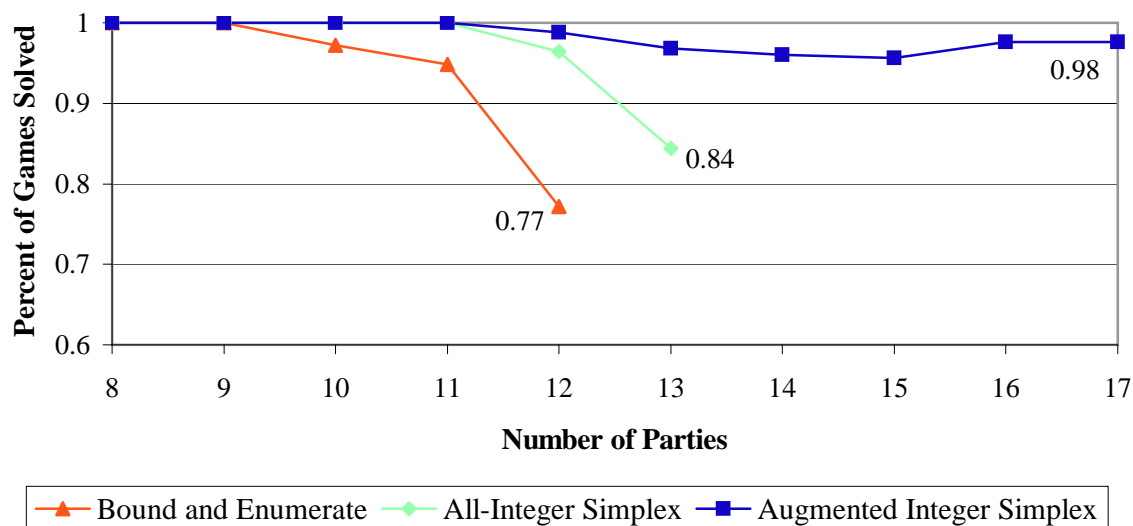


Figure 4-10: Percent of Games Solved by the Three Algorithms ($t \geq 50$)

⁵A similar rise would surely also be seen in bound and enumerate's effectiveness.

Chapter 5

Conclusions and Future Work

5.1 Conclusions

When developing these algorithms, important theoretical properties of minimum integer weights were discovered. First, interchangeable parties are useful in delineating weight differences between parties, but cannot guarantee that two parties have the same weight. Continuing along that path of research, one finds that minimum integer weights are non unique and that there is always a monotonic solution to a voting game. Thus, minimum integer weights will never fall into the trap of more seats lead to less power as at least one power index, Deegan-Packel [26], does (a comforting and intuitive solution).

The basic integer linear programming approach of branch and bound is not powerful enough to solve games of more than about eight players. Optimizing branch and bound by utilizing properties to the minimum integer problem results in the bound and enumerate algorithm. While this is a significant improvement, this algorithm starts to fail at about twelve players. The large advancement in this paper is to use Gomory's 1960 all-integer simplex algorithm to solve the optimization problem. Combining the two algorithms into augmented integer simplex yields a very powerful tool capable of solving over 90 percent of 14-player games in under ten minutes. The bottleneck for many games is now coalition enumeration—unthinkable progress only six months ago.

5.2 Improvements to the Algorithms

While great strides were made in developing algorithms to find the minimum integer weights of voting games, time constraints limited the amount of truly in-depth optimization that was completed on these algorithms. The first priority was to find the weights of European parliaments for related work [1]. That mission was completed successfully; the largest hurdle being the 1994 Italian election with 21 parties and over 10,000 coalitions (and 3,000 tying coalitions).¹ While further improvements were made after the empirical research, some sacrifices had to be made.

For instance, the fact that bound and enumerate is the slowest of the three algorithms has been apparent for some time, thus closer scrutiny was placed on the other algorithms. An unimplemented, direct approach to improve the “bound” portion of bound and enumerate would be to use an interior point method to solve the LP subproblems instead of the simplex method. Interior point methods are polynomial time algorithms [31], while simplex is only polynomial on average. These interior point processes, which do not usually return exact solutions, are a good fit for bound and enumerate since that algorithm does not require an exact solution from its subproblems.² The effect of this change greatly depends on how the “enumerate” portion of the algorithm scales on the size of the game. The relevant results in Figure 4.2 are equivocal, with increased complexity of games pushing the number of enumerations up, and the time limit and smaller weight reduction for larger games keeping the number smaller.

Also, more information could potentially be included in augmented integer simplex. Similar to bound and enumerate, the lower bounds for each variable would be found. This n -tuple, instead of the origin, would be useful as a starting point for the simplex algorithm. Exactly how to implement this change is beyond the mathemati-

¹Interestingly, the minimum integer weights were not the seat values. All parties with more than 50 seats had a minimum integer weight of one less than its number of seats.

²Since bound and enumerate takes the ceiling or floor of the subproblem solutions, an inexact (but accurate), solution would be sufficient in most cases. For cases in which the solution is an integer and the LP algorithm misses in the “wrong” direction, benefit of the doubt could be given to making the search space larger. Also note that polynomial LP algorithms that return exact solutions are available [31], but require more work and time.

cal ability of the author. But, the simplex algorithm seems dynamic enough to utilize this new information, and hopefully this additional information will speed up future implementations of the algorithm.

Putting this all into perspective, current researchers are comparing the empirical results from parliaments to proportional bargaining models (e.g., Morelli) and *formateur* models (e.g., Baron-Ferejohn), determining which model fits the data best. Minimum integer weights are proportional to seat share and correct for the relative weakness effect (a feature seat shares lack); thus, minimum integer weights can act as a stand in for the complex, open form proportional bargaining models. With algorithms such as augmented integer simplex, scholars can now more accurately contrast the different bargaining models.

Appendix A

Technical Efficiencies, Optimizations, and Details

Rational numbers and GCD calculation. Most programming languages do not have exact floating point arithmetic. Thus, each real number needs to be stored as a rational number (i.e., a pair of two integers). To simplify a rational number, the greatest common denominator (GCD) of the numerator and denominator must be calculated. It is efficient to determine the GCD of a rational number only when (1) the number is needed or (2) the numerator and denominator are so large that an overflow will occur in the near future.

Keeping a Library. To ensure that work is not lost, each solution is recorded in a library. The library is stored as a hashmap, with the minimal winning and unique tying rank coalitions as keys, and the minimum integer weights as values. Some non-vital data, such as the number of iterations that the respective algorithm needed to solve the problem, are also recorded. Currently, the library holds over 14,000 games.

Sorting the queue in branch and bound. A small optimization is to check whether a node is integral, and give it highest priority in the queue for its weight-level. Thus, one does not have to examine non-integral solutions of the same weight because the minimum integer weights would be found first.

On the Web. The algorithms have been incorporated into a Java Web Start application and has been placed on the World Wide Web at <http://web.mit.edu/aaronbs/www/thesis/>.

However, web addresses are ephemeral and no doubt the application will be relocated within six months. However, a Google search for “Minimum Integer Weight and Baron-Ferejohn Calculator” should produce the desired result.

Bibliography

- [1] Stephen Ansolabehere et al. Voting weights and formatuer advantages in coalition formation: Evidence from european paliaments, 1945-2000. Publication forthcoming, June 2003.
- [2] John F. Banzhaf III. Weighted voting doesn't work: A mathematical analysis. *Rutgers Law Review*, 19:317–43, 1965.
- [3] David P. Baron and John A. Ferejohn. Bargaining in legislatures. *American Political Science Review*, 83:1181–206, 1989.
- [4] Michael J. Best and Klaus Ritter. *Linear Programming: Active Set Analysis and Computer Programs*. Prentice-Hall Inc., New Jersey, 1985.
- [5] Vasken Bohossian and Jehoshua Bruck. On neural networks with minimal weights. Electronic Technical Report 5, California Institute of Technology, Department of Computation and Neural Systems and Electrical Engineering, June 1995.
- [6] Eric C. Browne and Mark Franklin. Aspects of coalition payoffs in european parliamentary democracies. *American Political Science Review*, 67:453–69, 1973.
- [7] Eric C. Browne and John P. Frendreis. Allocating coalition payoffs by conventional norm: An assessment of the evidence from cabinet coalition situations. *American Journal of Political Science*, 24:753–68, 1980.
- [8] W. Cook, C.R. Coullard, and Gy. Turan. On the complexity of cutting plane proofs. *Discrete Applied Mathematics*, 18:25–38, 1987.

- [9] W. Cook, A.M.H. Gerards, A. Schrijver, and E. Tardos. On the complexity of cutting plane proofs. *Mathematical Programming*, 34:251–264, 1986.
- [10] Robert T. Firla. *The Design of Exponential Neighborhoods - A Primal Approach to Integer Programming*. PhD thesis, Otto-von-Guericke-Universitat Magdeburg, Germany, April 2002.
- [11] William A. Gamson. A theory of coalition formation. *American Sociological Review*, 26:373–82, 1961.
- [12] Ralph E. Gomory. An all-integer programming algorithm. In *Industrial Scheduling*. Prentice-Hall, Inc., Englewood Cliffs, New Jersey.
- [13] Ralph E. Gomory. Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Mathematical Society*, 68:275–278, 1958.
- [14] Ralph E. Gomory. Early integer programming. In J. K. Lenstra, A. H. G. Rinnooy Kan, and A. Schrijver, editors, *History of Mathematical Programming: A Collection of Personal Reminiscences*. Elsevier Science, Amsterdam, The Netherlands, 1991.
- [15] J.R. Isabel. A class of majority games. *The Quarterly Journal of Mathematics*, 7(2):183–187, 1956.
- [16] William F. Lucas. Measuring power in weighted voting systems. In Stephen J. Brams, William F. Lucas, and Phillip D. Straffin, Jr., editors, *Political and Related Models*, volume 2 of *Modules in Applied Mathematics*, chapter 9. Springer-Verlag, New York, 1978.
- [17] Yasuko Matsui and Tomomi Matsui. NP-completeness for calculating power indices of weighted majority games. *Theoretical Computer Science*, 263, 1998.
- [18] Massimo Morelli. The demand bargaining set: General characterization and application to majority games. Technical Report 11, School of Science Working Paper Series, December 2001.

- [19] Massimo Morrelli. Demand competition and policy compromise in legislative bargaining. *American Political Science Review*, 93:809–20, 1999.
- [20] P. Narendra and K. Fukunaga. A branch and bound algorithm for feature subset selection. *IEEE Transactions on Computers*, 26:917–922, 1977.
- [21] Adras Sabo. An introduction to empty lattice simplices. In *Integer Programming and Combinatorial Optimization*, Graz, Austria, 1990. 7th International IPCO Conference.
- [22] Harvey M. Salkin and Kamlesh Mathur. *Foundations of Integer Programming*. North-Holland, New York, 1989.
- [23] Alexander Schrijver. *Theory of Linear and Integer Programming*. John Wiley & Sons, New York, 1986.
- [24] L.S. Shapley and Martin Shubik. A method for evaluating the distribution of power in a committee system. *American Political Science Review*, 48:787–92, 1954.
- [25] Daniel Solow. *Linear Programming: An Introduction to Finite Improvement Algorithms*. North-Holland, New York, 1984.
- [26] Aaron Strauss. An algorithm to calculate the minimum integer weights for arbitrary voting games. Bachelor’s thesis, Department of Political Science, Massachusetts Institute of Technology, June 2003.
- [27] James M. Snyder, Jr., Michael M. Ting, and Stephen Ansolabehere. Legislative bargaining under weighted voting. Publication Forthcoming, January 2003.
- [28] F.P. Vasilyev and A. Yu. Ivanitsky. *Indepth Analysis of Linear Programming*. Kluwer Academic Publishers, Dordrecht, The Netherlands, 2001.
- [29] John Von Neumann and Oskar Morgenstern. *Theory of Games and Economic Behavior*, chapter 10. Princeton University Press, Princeton, NJ, 1944.

- [30] Paul V. Warwick and James N. Druckman. Portfolio salience and the proportionality of payoffs in coalition governments. *British Journal Of Political Science*, 31, 2001.
- [31] Stephen J. Wright. *Primal-Dual Interior-Point Methods*. Society for Industrial and Applied Mathematics, Philadelphia, 1997.