THE PENNSYLVANIA STATE UNIVERSITY

SCHREYER HONORS COLLEGE

DEPARTMENT OF ECONOMICS

NON-SYMMETRIC STABLE SETS IN THE MAJORITY PILLAGE GAME

BOBAK PAKZAD-HURSON
Spring 2011

A thesis
submitted in partial fulfillment
of the requirements
for a baccalaureate degree
in Economics
with honors in Economics

Reviewed and approved* by the following:

James Jordan
Professor of Economics
Thesis Supervisor

David Shapiro
Professor of Economics
Honors Advisor

* Signatures are on file in the Schreyer Honors College

Abstract

The majority pillage game, introduced by Jordan (2006) is a model of Hobbesian anarchy in which a given number of players fight over a fixed amount of wealth, normalized to unity. Coalitions of varying sizes can form, and the power of coalitions is determined by their size and wealth. Therefore, power is endogenously defined within the game by means of a *power function*. More powerful coalitions have the ability to *pillage*, with certainty and without cost, weaker coalitions. Previous work on majority pillage games has focused on the stable set solution concept (von Neumann-Morgenstern solution) as an endogenous balance of power. This paper explores new results on the stable set in the case of four players. A programming model detailed within the paper suggests that stable sets do not exist – in other words, there is no endogenous balance of power for this game. A theoretical approach explores several properties of a stable set, and illustrates the possible areas which prevent the existence of stable sets. Finally, formal results for stable sets are given which can be generalized to the broader class of majority pillage games with more than four players.

Table of Contents

Acknowledgements

I would like to take this opportunity to thank my long time mentor and thesis adviser, Jim Jordan. Professor Jordan originally introduced me to pillage games, and patiently helped me learn the ropes. His support, ideas, and enthusiasm were invaluable during this experience.

I would also like to thank Professor David Shapiro for his efforts. Professor Shapiro's guidance and persistence made it nearly impossible for me to fall too far behind schedule on writing my thesis. His early revisions also made this thesis much easier for any readers to decipher.

Finally, I thank my great friend and roommate, Daniel Patrick Lubey, without whose programming expertise, this project may never have left the drawing board. For two years, Dan tolerated my frustration, and was always available to assist with difficult coding.

1. Literature Review

Several decades' worth of work in the field of cooperative game theory has led to the analysis included in the latter portions of this thesis. This literature review will discuss the beginnings of coalitional game theory, and the evolutionary process of this field which led to the creation and refinement of the majority pillage game. Any definitions or terminology used in this section will be defined and used in a non-technical manner. Following chapters will offer more formal definitions of necessary concepts.

1.1 Origins of Coalitional Game Theory

John von Neumann and Oskar Morgenstern in their 1944 classic, *Theory of Games and Economic Behavior*, lay out the foundation to what has become modern game theory. More specifically, von Neumann and Morgenstern introduce a class of cooperative games – games in which players must form coalitions to increase their utilities. Their most detailed (and longest lasting) game is now known as the classical majority game. The premise of this game is relatively simple. Suppose three people[1] are offered a possible $1, but can only achieve this amount if either two or all three of them band together to receive the money (this is a simplified description of the description of von Neumann and Morgenstern (1944, Chapter V)). If this majority coalition is formed, they must distribute the wealth between themselves in an agreeable manner. Indeed, there are an infinite number of possible ways that wealth can be distributed over this coalition of players. However, certain coalitions prefer specific allocations to others. If every player in the coalition strictly prefers wealth allocation x to allocation y, then allocation x is said to *dominate* allocation y (von Neumann and Morgenstern, 1944, p 37)[2]. Domination simultaneously allows for a solution concept, and makes such solutions difficult to find. However, von Neumann and Morgenstern present a concept for solving this dilemma through a set of stable allocations of this wealth, a notion which they aptly call the "solution." The solution of such a game (now known as a "stable set" or a "von Neumann-Morgenstern solution") is a set of wealth allocations which satisfy two conditions: none of the allocations in the stable set dominate one another, and every allocation not in the stable set is dominated by at least one allocation in the set (von Neumann and Morgenstern, 1944, p 40). Today, these two conditions are known as "internal stability" and "external stability" respectively (Lucas, 1992 p 547). A stable set for this game is the set containing {(½ , ½, 0) , (½, 0, ½) , (0 , ½, ½)} (Luce and Raiffa, 1957, p 200). Together, these three allocations dominate all other possible allocations.

While von Neumann and Morgenstern laid out a solution concept, they left the possibilities of existence and uniqueness of stable sets to future generations. The two remark, "We have previously mentioned, but purposely neglected to discuss, an important objection: That neither the existence nor the uniqueness of a solution S is … evident or established" for n-players (von Neumann and Morgenstern, 1944, p 42).

[1] This game was originally described for 3 players, and later expanded to include n-players in von Neumann and Morgenstern, 1944.

[2] For example, for a coalition containing players 1 and 2, (½, ½, 0) dominates (⅓, ⅓, ⅓) since both players gain a larger portion of the wealth (where player 1's wealth is denoted by the first number in the parentheses and player 2's wealth is denoted by the second number).

1.2 Present-day knowledge of Solution Concepts

The answers to existence and uniqueness have since been solved. Although it was long believed that stable sets exist for all characteristic function games (such as the majority game), Lucas (1968) disproves this notion. Lucas (1971, p 506) also demonstrates the lack of uniqueness of stable sets by giving examples of games with multiple stable sets. The lack of existence and uniqueness of stable sets are problems that von Neumann and Morgenstern considered, but did not address. In fact, they comment that "If it should turn out that our requirements concerning a solution S are, in any special case, unfulfillable – this would certainly necessitate a fundamental change in the theory" (von Neumann and Morgenstern, 1944, p 42). Lucas and Michaelis (1982) continue to critique the stable set for its undesirable properties; stable sets sometimes do not exist, and when they do exist, are often comprised of infinitely many allocations.

Two notable attempts at solving this issue relate to the majority pillage game. The first is the concept of the "core," introduced by Gillies (1959). In the majority game, the core, a stronger solution concept than the stable set, is the set of allocations that are completely undominated (Gillies, 1959). Indeed, the core is a subset of the stable set (Lucas, 1971). Unfortunately, as Lucas (1971, p 501) notes, "The main problem with the core is that it is frequently the empty set."

Economists have made attempts to link the functionality of stable sets to that of the core. Luce and Raiffa (1958), in their discussion of properties of stable sets, note that players often do not have any incentive to attempt to leave the stable set. Chwe (1994, p 300) clarifies this notion by defining a "farsighted" property which can be generalized to stable sets in which "a coalition considers the possibility that once it acts, another coalition might react, a third coalition might in turn react, and so on, without limit." These analyses are close to linking the core to "farsighted" stable sets, in the sense that forward-looking coalitions may elect not to try to depart from a stable set allocation. However, Luce and Raiffa's notion is not explored in enough depth to make this claim, and Chwe's analysis is not able to directly tie farsightedness to a lack of motivation for coalitions to depart from an allocation contained in the stable set.

The second attempt revolved around refining the definition of a solution, not redefining it. The restriction imposed upon stable sets is often that of symmetry. A stable set is said to be symmetric if it contains all permutations of every allocation in the set (Lucas, 1992, p 573). Taking the numerical example given above, this means that if (½ , ½, 0) is in the stable set, then (½, 0, ½) and (0 , ½, ½) must also be in the stable set to ensure that the solution is symmetric[3]. By strictly analyzing symmetric stable sets, a general formulation for odd numbers of players is found. Indeed, Bott (1953) shows that there is a unique symmetric stable set for the majority game for odd numbers of players[4]. Not only does the result hold for all odd numbers of players, but its

[3] (Luce and Raiffa 1957, p 200) give an alternate way to think about this concept. Suppose (½ , ½, 0) is in the set. Permute the numbering of each of the players, and assign them the new values of wealth. The result is the same: (½, 0, ½) and (0 , ½, ½) must be a part of the stable set to make it symmetric.

[4] For the (2k +1)-person game, the unique symmetric stable set is $V_B^n = \langle \left(\frac{1}{k+1}\right), \dots, \left(\frac{1}{k+1}\right), 0, \dots, 0 \rangle$ where each allocation has (k+1) strictly positive components (Lucas, et. al. 1982, p 118). For example, the 5 person majority game (k=2) has a unique stable set containing all permutations of (⅓, ⅓, ⅓, 0, 0).

formulation is also extremely simple. This simplicity is further demonstrated by the fact that no such formulation has been made for the majority game with even numbers of players. Lucas (1968) proves that there is no symmetric stable set for $n = 8$ players. Even with 4 players (the smallest even number of players for which the stable set has meaning as a solution concept) all possible stable sets have yet to be discovered (Lucas, 1971, p 505).

1.3 The Majority Pillage Game

Jordan (2006) introduced a new class of game known as the pillage game, designed to model a Hobbesian anarchy. Pillage games are "represented by a power function that specifies the power of each coalition as depending at least partly on the wealth of its members" (Jordan, 2006, p 27). More powerful coalitions are able to strip weaker ones of all or part of their wealth. Among these classes of pillage games, the majority pillage game is introduced in Jordan and Obadia (2007). While similar to the classical majority game, by adding wealth into the power function for the majority game, ties can now be broken; a two-player coalition can be stripped of its wealth by a richer two-player coalition.

The two solution concepts in this game parallel those in the classical majority game. The core is a set of allocations such that each player can defend his own wealth by force. Therefore, any allocation in the core is undominated by any other allocation. Unfortunately, once the number of players expands beyond two, the core becomes the empty set (Jordan and Obadia, 2007, p 5). In essence, there are now too many players for any one of them to be able to fend off all of the others by force. Therefore, the search for solutions in the majority pillage game must be limited to stable sets.

Stable sets are analogous to a balance of power. While no single player can defend against all others by herself, a majority coalition can band together to preserve its dominance. Due to the structure of these pillage games, Jordan is able to link Chwe's farsighted property to stable sets. Jordan (2006) proves that if players hold common future expectations, stable sets act as a farsighted core – every allocation outside of the stable set is dominated in expectation. In stable set allocations, therefore, there is no incentive for any new coalitions to form to exit the set.

The complication of the classical majority game by adding wealth into the mix has, in fact, simplified the results. Compared to the classical majority game where stable sets could be comprised of infinitely many allocations, the stable sets in the majority pillage game are comprised of "at most finitely many allocations" (Jordan, 2006, 33). Furthermore, all symmetric stable sets have been discovered: for odd numbers of players, the unique symmetric stable set is identical to the one shown in Bott (1953), while no non-empty symmetric stable set exists for even numbers of players (Jordan and Obadia, 2007, p 8).

1.4 Anarchy

The pillage game is not the first model of Hobbesian anarchy. Several other authors have attempted to answer the question: Can anarchy lead to cooperation? Ghiselin (1978), while not necessarily relating anarchy to cooperation, finds that anarchy leads to a natural ordering, where implicit cooperation is possible. Expanding upon Ghiselin, Hirshleifer's (1995, p 26) model of

anarchy finds that a "spontaneous order" can form under certain conditions. More closely related to Jordan's findings, Skaperdas (1992, p 733) finds that in anarchy "the agents cooperate when conflict is ineffective". This confirms Jordan's findings of the farsighted stability offered by a stable set in the majority pillage game. Because of this farsighted stability, any act of pillage will necessarily harm some of the players who undertake it, making conflict extremely ineffective for these individuals. Hirshleifer also finds that the number of players (or agents) determines fighting intensity; specifically, "as N[5] grows exogenously, equilibrium fighting intensities… rise" (Hirshleifer, 1995, p 47).

This partially confirms Jordan's findings. As N grows exogenously beyond 2, the core becomes empty. This can be viewed as an increase in fighting, since no allocation of wealth is immune to pillage any longer. However, as discussed above, Jordan and Obadia find unique, symmetric stable sets for odd numbers of players. The intensity of "fighting" within these stable sets, therefore, does not appear to vary when the number of players is 5 or 55. This seems to contradict Hirshleifer's conclusion about fighting. Furthermore, analysis conducted by this author (which is included in latter portions of this thesis) indicate that there is no stable set in the majority pillage game for 4 players. With 4 players, there may be just too much "fighting" as Hirshleifer puts it for there to be any "equilibrium." And yet there is seemingly less fighting when the number of players increases to 5. This could be another contradiction to Hirshleifer's findings. However, neither of the analyses offered by Jordan nor by this author specifically relates the number of players to "fighting intensities" in the stable set. Perhaps future authors will clarify this point.

With this possible exception, the majority pillage game seems to conform to previous findings on cooperation under anarchy.

1.5 Recent findings involving stable sets

Some recent findings regarding the maximum size of stable sets in the pillage game have arisen. Kerber and Rowat (2009) place an upper bound on the maximum size (number of allocations contained in the set) of stable sets for all pillage games (not necessarily majority pillage games). Saxton (2009, p 14) offers an even clearer bound: the maximum size of a stable set for a 4 person pillage game is 3^{15}; until this author's analysis, Saxton's bound was the best bound for pillage games. Saxton's results are contrasted significantly in Section 5, which empirically indicates a much smaller maximum bound for the four person majority pillage game, and is shown by construction to be several orders of magnitude lower than Saxton's bound in Section 3.

[5] Where N represents the number of players.

2. Pillage games

Environment:

The environment for pillage games is comprised of a fixed amount of total wealth (which is normalized to unity), which can be allocated among a fixed number of players.

2.1 Definitions: The set of players is the finite set $I = \{1, \ldots, n\}$, where $n \geq 2$. Subsets are called *coalitions*. The set of *allocations* is the set $A = \{w \in \mathbb{R}^n \mid w_i \geq 0 \text{ for each } i, \text{ and } \sum_i w_i = 1\}$. Note that A is a simplex in $\mathbb{R}^n$.

Coalitions are assigned a numerical value of *power* which is determined by a *power function.* More powerful coalitions are able to *pillage* weaker ones, which means they are costlessly able to acquire any proportion of the weaker coalition's wealth. There are several axioms which a power function must follow:

> "The power of a coalition does not decrease if new members are added (p.1) or if the wealth of some members is increased without decreasing the wealth of other members (p.2). Increasing the wealth of every member increases the power of the coalition (p.3). This requirement excludes power functions that are independent of wealth, such as those that depend on coalitional size alone." (Jordan, 2006 p. 29)

Formally, these axioms are presented in the following definition.

2.2 Definitions: The *power function* is a function $\pi: 2^I \times A \to \mathbb{R}$ satisfying
(p.1) if $C \subset C'$ then $\pi(C', w) \geq \pi(C, w)$ for all w;
(p.2) if $w'_i \geq w_i$ for all $i \in C$ then $\pi(C, w') \geq \pi(C, w)$; and
(p.3) if $C \neq \emptyset$ and $w'_i > w_i$ for all $i \in C$ then $\pi(C, w') > \pi(C, w)$.

Domination Relation:

An allocation w' *dominates* an allocation w, written $w' \succ w$, if

$\pi(W,w) > \pi(L,w)$,
where $W = \{i \mid w'_i > w_i\}$ and $L = \{i \mid w'_i < w_i\}$

In plain language, domination is a transition mechanism between allocations. In the definition above, coalition W is more powerful at allocation w than coalition L. Since the members of W prefer allocation w' and are more powerful, they are able to move from allocation w to allocation w'.

This thesis will focus on a particular pillage game known as the *majority pillage game*.

2.3 Definition: A *majority pillage game* is a pillage game with a power function of the form

$$\pi(C, w) = v \cdot \#C + \sum_{i \in C} w_i \text{ for some } v > 1.$$

Since the total wealth of all players is normalized to 1, $v > 1$ implies that wealth can only serve as a tie-breaker; wealth is only a factor in determining dominance if the coalition in favor of a transition and the coalition opposing it are of equal size.

Solution Concepts:

The following definitions follow from von Neumann and Morgenstern (1947) and Lucas (1992).

2.4 Definitions: The *core* is the set of undominated allocations. A set S of allocations is a *stable set* if it satisfies

(IS) (internal stability) no element of S is dominated by an element of S; and
(ES) (external stability) every element of $A \setminus S$ is dominated by some element of S.

3. Internal Stability

Internal stability has limits. Jordan (2006) proves that internally stable sets contain at most finitely many allocations. Because of this, every internally stable set has upper bound on the number of elements it can contain. [6]

3.1 Definition: An internally stable set S' is said to be a *maximal internally stable set* if S' is internally stable and there exists no $w' \notin S'$ such that $S = \{w'\} \cup S'$ is an internally stable set.

In other words, a set is a maximal internally stable set if the addition of any other wealth allocation to the set will violate internal stability. Note that just because an internally stable set is *maximal* does not mean it contains the *maximum* number of elements possible in an internally stable set.

3.2 Proposition: For any maximal internally stable set $S' = \{w_1, w_2, \dots, w_n\}$ which is not stable, a stable set S must (1) dominate at least one allocation $w \in S'$. Furthermore, set S must (2) contain every allocation in S' which it fails to dominate.

Proof of 3.2 Proposition: To prove (1), suppose by way of contradiction that for all $w_i \in S'$ there is no $w'_k \in S$ which dominates w_i. Since S is assumed to be stable, it is both internally and externally stable. Therefore, since S does not dominate w_i, w_i must be in S. Since this logic holds for all $1 \leq j \leq n$, $S = \{w_1, w_2, \dots, w_n\} \cup \{w'_1, \dots, w'_k\}$. However, since $S' = \{w_1, w_2, \dots, w_n\}$ is a maximal internally stable set, $\{w_1, w_2, \dots, w_n\} \cup \{w'_1, \dots, w'_k\}$ cannot be internally stable. This contradicts the assumption that S is stable.

To prove (2), suppose by way of contradiction that S does not dominate some allocation $w_i \in S'$ and $w_i \notin S$. Since S is assumed to be stable, it is also externally stable. However, S fails to dominate $w_i \notin S$, which contradicts external stability, which completes the proof. ■

The notion of symmetry also plays a role in internally stable sets. Symmetry implies that if a particular allocation w is in a set, every player permutation of w is also in the set.

3.3 Definitions: A *player permutation* is a one-to-one and onto function $t: I \to I$. Let T be the set of all player permutations. For each t, the function $\sigma_t: A \to A$ is defined as $\sigma_t(w) = (w_{t(1)}, \dots, w_{t(n)})$. For arbitrary set of allocations S, $\sigma(S) = \bigcup_{t \in T} \sigma_t(S)$.

3.4 Definition: A set of allocations S is *symmetric* if $\sigma(S) = S$. In other words, a set is symmetric if it contains every permutation of each allocation it contains.

As a means of convenience, it is often useful to "rank" the wealth levels in an allocation in decreasing order. This results in the following definition.

3.5 Definition: An n-player wealth allocation w is said to be in *standard form* if $w_1 \geq w_2 \geq w_3 \geq \dots \geq w_n$.

Note that in a standard form allocation, the first $\frac{n}{2}$ players control at least half of the wealth.

[6] This is the topic of Section 5

3.6 Proposition: Let $S^\circ = \sigma(½,½,0,0)$. Let $w' \notin S^\circ$ such that $S = S^\circ \cup \{w'\}$ is internally stable. Then $w' = (¼, ¼, ¼, ¼)$ and S is a maximal internally stable set.

Proof of 3.6 Proposition: Since S° is symmetric, we can assume without loss of generality that any allocation $w' \notin S^\circ$ is in standard form. It is easy to check that $S = S^\circ \cup \{ (¼, ¼, ¼, ¼) \}$ is internally stable. The remainder of the proof will be divided into the following 4 cases which will show that any $S' = \{w'\} \cup S$ is not internally stable for any $w' \notin S$.

Case 1: $w_1' < ½$

Since $w' \notin S$, $w_1' + w_2' > ½$. Therefore $w' \prec (½,½,0,0)$ which violates the internal stability of S.

Case 2: $w_1' \geq ½$, $w_4' = 0$

Since $w' \notin S$, we know $w_1' \geq ½$ and $w_2' < ½$. Therefore $(0,½,½,0) \succ w'$ which violates the internal stability of S.

Case 3: $w_1' \geq ½$, $w_2' \geq ¼$, $w_4' > 0$

In this case $w_3' > 0$. Therefore $w' \succ (½,½,0,0)$ which violates the internal stability of S.

Case 4: $w_1' \geq ½$, $w_2' < ¼$, $w_4' > 0$

Since $w' \notin S$, we know $¼ > w_2'$, w_3', $w_4' > 0$. Therefore $w' \succ (½,½,0,0)$ which violates the internal stability of S. Also note that $(¼, ¼, ¼, ¼) \succ w'$.

As can be seen by the above cases, any $S = S^\circ \cup \{w'\}$ cannot be internally stable for $w' \neq (¼, ¼, ¼, ¼)$, as the addition of any new allocation violates internal stability. Since S is internally stable, therefore it is a maximal internally stable set. ■

3.7 Remark: As can be seen in Case 3, the set S fails to dominate every allocation $w' \notin S$. Therefore, S is not externally stable.

3.8 Corollary: Any stable set S^* must contain at least one allocation w which dominates at least one permutation of $(½,½,0,0)$. Furthermore, this allocations has $w_j \geq ½$ for some $j = \{1,2,3,4\}$.

Proof of 3.8 Corollary: This proof easily follows from 3.2 Proposition and Case 3 of 3.6 Proposition.

3.9 Proposition: If a set S is stable, then S is also a maximal internally stable set.

Proof of 3.9 Proposition: Suppose by way of contradiction that S is stable and not a maximal internally stable set. Since S is stable, we know that it is externally stable. Since we are assuming that S is not a maximal internally stable set, then there is some $w' \notin S$ such that $S^* = \{w'\} \cup S$ is internally stable. Then there is no $w \in S$ such that w dominates w'. Therefore, S is not externally stable. This contradicts the assumption that S is a stable set. ■

The following propositions are necessary for the construction of 3.15 Proposition, the main result of this section.

3.10 Proposition: For any two allocations w' and w, if $\#\{i|w_i' \neq w_i\} = odd$ then either $w' \succ w$ or $w \succ w'$.

Proof of 3.10 Proposition: Since $\sum_{i\in I} w_i = \sum_{i\in I} w_i' = 1$, if $\#\{i|w_i' \neq w_i\} = odd$, then $\#\{i|w_i' > w_i\} \gtrless \#\{i|w_i' < w_i\}$. By the power function defining the majority pillage game, if $\#\{i|w_i' > w_i\} > \#\{i|w_i' < w_i\}$ then $w' \succ w$. If $\#\{i|w_i' > w_i\} < \#\{i|w_i' < w_i\}$ then $w \succ w'$. ■

3.11 Example: In the n =4 majority pillage game, if $\#\{i|w_i' = w_i\} = 1$, then either $w' \succ w$ or $w \succ w'$.

3.12 Proposition : For any two allocations w' and w, if $w'_i > w_i > ½$ for some $i \in$ I, then either $w' \succ w$ or $w \succ w'$. If $w_i > 0$ for all $i \in$ I then the previous inequality can be relaxed to $w'_i > w_i \geq ½$.

Proof of 3.12 Proposition:

Let coalition $C = \{i|w_i' > w_i\}$ and let $M = \{i|w_i' < w_i\}$. In other words, let C be comprised of the players of the players who prefer allocation w' to w and let coalition M be comprised of all players who prefer allocation w to w'. The proof will be divided into two cases:

Case 1: Let $\#C \neq \#M$

Since w and w' are arbitrary allocations, suppose without any generality loss that $\#C > \#M$ and the members of C prefer allocation w' to w. Then $\pi(C, w) > \pi(M, w)$, meaning that $w \succ w'$.

Case 2: Let $\#C = \#M$

Since w and w' are arbitrary allocations, without loss of generality let $w_1 > w'_1 > ½$. This implies that {player 1} $\in M$. Since $\#C = \#M$ and $\sum_{i\in M} w_i' > \frac{1}{2} > \sum_{i\in C} w_i'$, $\pi(M, w') > \pi(C, w')$, meaning that $w \succ w'$.

If $w_1 > w'_1 = ½$ and all players have strictly positive wealth at w, we know {player 1} $\in M$. Therefore, any coalition involving player 1 has more than half of the wealth. The rest of the proof is identical. ■

3.13 Remark: Let S be a stable set. From 3.8 Corollary, we know there is an allocation $w \in S$ such that $w_i \geq ½$ for one $i \in I$. Then for any $w^* \in S$, either $w_i^* = w_i$ or $w_i^* < ½$.

3.14 Proposition: Take a standard form allocation w where $w_1 \geq ½$, $w_2 < ½$, and a permutation w^*. If $\#\{i|w_i^* > w_i\} = \#\{i|w_i^* < w_i\} = {}^{n}/_{2}$ then neither w nor w^* dominates the other.

Proof of Proposition 3.14: Let coalition $C = \{i|w_i^* > w_i\}$ and coalition $L = \{i|w_i^* < w_i\}$. Since $\#\{i|w_i^* > w_i\} = \#\{i|w_i^* < w_i\} = {}^{n}/_{2}$ we know that $½ > w_2 \geq w_1^*$. Therefore, player 1 prefers allocation w to allocation w^*. Since $\#C = \#L$ and $\sum_{i\in C} w_i > ½$, $w \prec w^*$.

Since w^* is a permutation of w, $w_1 = w_i^*$ for some $i \in I$. Without loss of generality, suppose $w_1 = w_2^*$. Therefore, we know that $w_2^* \geq ½ > w_2$. This tells us that player 2 prefers allocation w^* to w. Moreover, since $\#C = \#L$ and $\sum_{i \in L} w_i^* > ½$, $w \succ w^*$. ■

The following proposition gives an upper bound for the number of allocations in an internally stable set in the four person majority pillage game. It is important to note that while this bound is several orders of magnitude tighter than the bound presented in Saxton (2009) for pillage games, it is not necessarily a best possible bound. Indeed, certain constraints (such as the constraint demonstrated in 3.13 Remark) are omitted in the construction of this bound.

Proposition 3.15: Let S be an internally stable set in the n = 4 majority pillage game. Then $\#S \leq 3^8$.

Proof of 3.15 Proposition:

3.10 Proposition implies that for any two allocations $w', w \in S$, $\#\{i | w_i' = w_i\} = 0$ or 2. We will examine each case separately, and then combine the results.

3.16 Lemma: Let S' be an internally stable set in the n = 4 majority pillage game. Further suppose that any two allocations $w', w \in S'$, $\#\{i | w_i' = w_i\} = 0$. Then $\#S \leq 3^2$.

Proof:

Since $\#\{i | w_i' = w_i\} = 0$ we can order the allocations by player 1's wealth. In particular, denote the allocations in S as w^1, w^2, w^3 … where $w_1^1 < w_1^2 < w_1^3 < \ldots$ Define for $k = 1,2,3, \ldots$

$$W^k = \{i | w_i^{k+1} > w_i^k\}$$
$$L^k = \{i | w_i^{k+1} < w_i^k\}$$

These two sets can be thought of as the "winners" and the "losers" when moving from one allocation in set S to the next. By the way we constructed S, note that player 1 is always a "winner". In order to preserve internal stability, and since no player has a common wealth level in any two allocations, the two of the remaining three players must be losers. Each allocation can be denoted as components in a four-dimensional direction vector d, defined as follows:

$$d_i = \begin{cases} 1 & \text{if } i \in W, \\ -1 & \text{if } i \in L, \\ 0 & \text{otherwise.} \end{cases}$$

Saxton (2009) proves that if there exists some $V \subset S$ with $\#V > 3$ such that for all $w \in V$, d_i is constant for all i, then S is not internally stable. We will refer to this as the monotonocity constraint. Since $d_i = 1$ for all $w \in S$, there are three components of the d vector that can vary. Since $d_i = -1$ for exactly two players for each w, there are $\binom{3}{2} = 3$ positions in which to place these two players in the vector. These correspond to three unique vectors, d^1, d^2, and d^3, where:

$$d^1 = (1,1,-1,-1)$$

$$d^2 = (1,-1,1,-1)$$
$$d^3 = (1,-1,-1,1)$$

In order to abide by the monotonocity constraint, there can be at most 3 vectors for each d^1, d^2, d^3. Then there can be at most $3 \cdot 3 = 9$ allocations in *S.*

3.17 Lemma: Let S' be an internally stable set in the n = 4 majority pillage game. Further suppose that any two allocations $w', w \in S'$, $\#\{i|w_i' = w_i\} = 2$. Then $\#S' \leq 3^6$.

Proof: We will once again use the notation introduced in the proof of 3.16 Lemma. Note that in this case there are $\binom{4}{2} = 6$ ways to choose the two players where $d_i = 0$. These correspond to 6 direction vectors. In order to preserve internal stability, we must have $\#W = \#L$. Therefore, $\#W = \#L = 1$. Internally stability implies that there can be at most 3 wealth allocations $w_{d_j}^1 w_{d_j}^2, w_{d_j}^3$ corresponding to each direction vector d_j. We will recursively define the number of elements which can be in S'. Let x^j represent the maximum number of allocations that can be in S' after considering the j^{th} direction d_j for $j = \{1,2, \dots ,6\}$. Let y^jrepresent the maximum number of allocations that can be added to S' after considering the j^{th} direction d_j for $j = \{1,2, \dots ,6\}$. For j=1, we have shown that y^1=$x^1 = 3$. Following the same pattern of logic, when j=2, for each of the $x^1 = 3$ allocations, we can add 2 allocations. This implies that $y^2 = 2 \cdot x^1$. In general, we have

$$y^j = 2 \cdot x^{j-1}$$

Since $x^j = x^{j-1} + y^j$, substituting for y^jwe find that

$$x^j = 3 \cdot x^{j-1} = 3^{j-1} \cdot x^1.$$

Letting $x^1 = 3$ and j=6, we get the desired result that x^6=3^6.

Proof of 3.15 Proposition:

Select an allocation corresponding to a direction found in 3.16 Lemma. By 3.17 Lemma, we can add on at most $3^6 - 1$ allocations to form a set V^1 such that for each $w, w' \in V^1$ $\#\{i|w_i' = w_i\} = 2$. Repeat this for each of the 3^2 allocations in Lemma 3.16. This implies that

$$\#S \leq \#\bigcup_{l=1}^{3^2} V^l = 3^2 \cdot 3^6 = 3^8.$$

■

Saxton (2009) bounds internally stable sets in four person pillage games at just under one million allocations. Saxton's bound applies to all pillage games, however, 3.15 Proposition tells us that Saxton's bound is at least 144 times too large.[7]

[7] This topic is explored further in Section 5. Also note that the results in Section 5 indicate that the bound in 3.15 Proposition is not necessarily as tight as possible.

4. External Stability

Unlike internally stable sets, externally stable sets can be infinite. Indeed, the largest (and most trivial) example of an externally stable set it the entire simplex A. Adding elements to any set, makes the set in question "more" externally stable, in the sense that the set will dominate more allocations outside the set with the additions. Therefore, it makes sense to think of the existence of a lower bound for the size of externally stable sets, analogously to the upper bound for the size of internally stable sets. If the supremum of the size of internally stable sets is greater than the infemum for the size of externally stable sets, it may be possible for a stable set to exist. Note that the work in the section refers specifically to the majority pillage game with 4 players.

We know from Corollary 3.8 that in any stable set S, there is at least one allocation w such that $w_i \geq$ ½ for one $i \in I$. We can permute set S to let w be in standard form, and take {player 2, player 3} $= C$ and {player 1} $= L$. From the first axiom of power functions, it is necessarily the case that $\pi(C, w) > \pi(L, w)$. Therefore, coalition C can take any amount of wealth $w \in (0, w_1]$ from coalition L in a pillage. Since the pillaging coalition can take an arbitrary proportion of player 1's wealth, the differences between w and w' can be arbitrarily small. The resulting allocation w' dominates allocation w such that $w_1 > w_1'$, $w_2' > w_2$, $w_3' > w_3$ and $w_4' = w_4$. If S is externally stable, then there must be $w^* \in S$ such that $w^* \succ w'$ where $w^* \nsucc w$ and $w^* \nprec w$. The following proposition shows the form this allocation w^* must take.

4.1 Proposition: If a stable set S exists in the 4-player majority pillage game, it must include at least one allocation w^* in one of the following forms:

(w_1, w_2, a, b) where $a < b$
(w_1, c, w_3, d) where $c < d$
(w_1, e, f, w_4) where $e < f$

4.2 Lemma: $w_1 = w_1^*$.
Proof:

Case 1: Suppose by way of contradiction that $w_1 < w_1^*$.
Then $\#\{i|w_i^* > w_i\} \geq 2$ or $\#\{i|w_i^* > w_i\} = 1$. If $\#\{i|w_i^* > w_i\} \geq 2$ then $w^* \succ w$ since $w_1^* > ½$. If $\#\{i|w_i^* > w_i\} = 1$ and $\#\{i|w_i^* < w_i\} = 1$ then $w^* \succ w$ since $w_1^* > ½$. Otherwise, $w \succ w^*$.

Therefore, $w_1 \geq w_1^*$.

Case 2: Suppose by way of contradiction that $w_1 > w_1^* > w_2^*$.
Then $\#\{i|w_i^* > w_i\} = 3$ or $\#\{i|w_i^* > w_i\} = 2$ or $\#\{i|w_i^* > w_i\} = 1$.
If $\#\{i|w_i^* > w_i\} = 3$ then $w^* \succ w$.
If $\#\{i|w_i^* > w_i\} = 2$ then

a) If $\#\{i|w_i^* = w_i\} = 1$ then $w^* \succ w$ by 4.12 Proposition.
b) If $\#\{i|w_i^* = w_i\} = 0$ and $\sum_{\{i \in I|w_i^* > w_i\}} w_i^* > ½$ then $w^* \succ w$.
c) If $\#\{i|w_i^* = w_i\} = 0$ and $\sum_{\{i \in I|w_i^* > w_i\}} w_i^* < ½$ then $w \succ w^*$.

d) If $\#\{i|w_i^* = w_i\} = 0$ and $\sum_{\{i\in I|w_i^* > w_i\}} w_i^* = ½$ then $w_1^* \leq ½$. Since w_1' can be arbitrarily close to ½, we can assume that $w_1^* < w_1'$. This implies that $\#\{i|w_i' > w_i^*\} \geq 2$ and $\sum_{\{i\in I|w_i' > w_i^*\}} w_i' \geq ½$. Therefore, $w^* \not\succ w'$ and set S cannot be externally stable.

If $\#\{i|w_i^* > w_i\} = 1$ then $w \succ w^*$ since $w_1 > w_2^*$.

Therefore, $w_1^* \geq w_1$ or $w_1^* \leq w_2^*$.

Case 3: Suppose by way of contradiction that $w_1^* \leq w_2^*$ and that $w^* \succ w'$
In order for w^* to dominate w', $\#\{i|w_i^* > w_i'\} = 3$ or $\#\{i|w_i^* > w_i'\} = 2$ and $\sum_{i\in\{I|w_i^* > w_i'\}} w_i' > ½$. Either way, $\pi(C, w') > \pi(L, w')$ where $w_i^* \geq w_i'$ only for all $i \in C$. Since the differences between w' and w can be arbitrarily small, this implies $\pi(C, w) > \pi(L, w)$. Therefore, $w^* \succ w$ which contradicts the internal stability of S.

Therefore, $w_1^* > w_2^*$.

From Case 3, $w_1^* > w_2^*$. Therefore, putting Cases 1 and 2 together, it is know that $w_1 \geq w_1^*$ and $w_1^* \geq w_1$. Therefore, $w_1 = w_1^*$. ■

4.3 Lemma: $\#\{i|w_i^* = w_i\} = 2$

Proof:

Suppose by way of contradiction that $\#\{i|w_i^* = w_i\} \neq 2$. Since $w_1 = w_1^*$, this implies that $\#\{i|w_i^* = w_i\} = 1$. By 4.12 Proposition this would mean that $w^* \succ w$ or $w \succ w^*$. Either situation contradicts the internal stability of S. Therefore, $\#\{i|w_i^* = w_i\} = 2$. ■

Proof of 4.1 Proposition:

Since there is $w' \notin S$ which dominates $w \in S$, there must also exist $w^* \in S$ such that $w^* \succ w'$ where $w^* \not\succ w$ and $w^* \not\prec w$ to preserve internal stability. From 5.2 and 5.3 Lemmas, w^* must appear in one of the following forms:

(w_1, w_2, a, b) where $a < b$ or else $w^* \succ w$.
(w_1, c, w_3, d) where $c < d$ or else $w^* \succ w$.
(w_1, e, f, w_4) where $e < f$ or else $w^* \succ w$. ■

Again, we know from Corollary 3.8 that in any stable set S, there is at least one allocation w such that $w_i \geq ½$ for one $i \in I$. We can permute set S to let w be in standard form, and take {player 1} = C and {player 2} = L. From the first axiom of power functions, it is necessarily the case that $\pi(C, w) > \pi(L, w)$. Therefore, coalition C can take any amount of wealth $w \in (0, w_2]$ coalition L in a pillage. The resulting allocation w' dominates allocation w such that $w_1' > w_1$, $w_2 > w_2'$, $w_3' = w_3$ and $w_4' = w_4$. Since the player 1 can take an arbitrary proportion of player 2's wealth, the differences between w and w' can be arbitrarily small. If S is externally stable, then there must be $w^\wedge \in S$ such that $w^\wedge \succ w'$ where $w^\wedge \not\succ w$ and $w^\wedge \not\prec w$. The following proposition details this allocation $w^\wedge$.

4.4 Proposition: In any stable set S, there must exist at least one allocation $w^\wedge$ from each of the following groups:

Group 1:

(a, w_2, w_2, b) where $a < b$
(c, w_2, d, w_4) where $c < d$

Group 2:

(e, f, w_3, w_4) where $e < f$
(g, w_2, w_3, h) where $g < h$

Group 3:

(i, w_2, j, w_4) where $i < j$
(k, l, w_3, w_4) where $k < l$

Without loss of generality, this proof will show that there must be at least one allocation $w^\wedge$ from Group 1. The proofs for Groups 2 and 3 can be attained by permuting Lemmas 4.5 and 4.6 to apply to players 3 and 4.

4.5 Lemma: $w_2 = \hat{w_2}$
Proof:

Case 1: Suppose by way of contradiction that $\hat{w_2} < w_2$.
Then there exists w' such that $\hat{w_2} < w_2' < w_2$. Therefore, it suffices to show that $\hat{w_2} < w_2'$ implies that $w^\wedge \notin S$.

In order for $w^\wedge$ to dominate w', $\#\{i | \hat{w_i} > w_i'\} = 3$,
$\#\{i | \hat{w_i} > w_i'\} = 2$ and $\sum_{i \in \{I | \hat{w_i} > w_i'\}} w_i' > ½$, or
$\hat{w_1} > w_1'$ and $\hat{w_3} = w_3'$ and $\hat{w_4} = w_4'$.
Then $\pi(C, w') > \pi(L, w')$ where $\hat{w_i} \geq w_i'$ only for all $i \in C$. Since the differences between w' and w can be arbitrarily small, this implies $\pi(C, w) > \pi(L, w)$.
Therefore, $w^\wedge \succ w$ which contradicts the internal stability of S.

Therefore, $w_2 \leq \hat{w_2}$.

Case 2: Suppose by way of contradiction that $\hat{w_2} > w_2$.

a) If $\hat{w_1} > ½$ and $\hat{w_1} \neq w_1$ then either $w^\wedge \succ w$ or $w \succ w^\wedge$ by 4.12 Proposition, which violates the internal stability of S.
b) If $½ \leq \hat{w_1} = w_1 < w_1'$, then $\sum_{i=3}^{4} \hat{w_i} < \sum_{i=3}^{4} w_i'$ since $\hat{w_2} > w_2 > w_2'$. Therefore, $w^\wedge \not\succ w'$.
c) If $\hat{w_1} < ½$, then $\hat{w_3} \geq w_3'$ and $\hat{w_4} \geq w_4'$ with at least one strict inequality for $w^\wedge$ to dominate w'. Since $w_3 = w_3'$, $w_4 = w_4'$ and $\hat{w_2} > w_2$ by assumption, then $w^\wedge \succ w$, which violates the internal stability of S.

Therefore, $w_2 \geq \hat{w_2}$.

From Cases 1 and 2, we know that $w_2 \leq \hat{w_2}$ and $w_2 \geq \hat{w_2}$, which imply that $w_2 = \hat{w_2}$. ■

4.6 Lemma: Either $w_3 = \hat{w_3}$ or $w_4 = \hat{w_4}$.
Proof: Since $w_2 = \hat{w_2}$ by 5.5 Lemma, 4.12 Proposition tells us that 2 players must have the same wealth in $w^\wedge$ and w. Suppose by way of contradiction that $\hat{w_1} = w_1$. Since $w_2 = \hat{w_2}$ and there is $w' = (w_1 + \varepsilon, w_2 - \varepsilon, w_3, w_4)$ for $\varepsilon \in (0, w_2]$, if $\hat{w_1} = w_1$, then $\sum_{i=3}^{4} \hat{w_i} = \sum_{i=3}^{4} w_i'$. Assuming $w^\wedge \neq w$, Then either $\hat{w_3} > w_3$ *or* $\hat{w_4} > w_4$> Either way, $\#\{i \mid \hat{w_i} < w_i'\} = 2$ and since $w_1' > w_1 = \hat{w_1} \geq ½$, $w^\wedge \nsucc w'$. Since $w' \succ w$, this contradicts the external stability of S. ■

Proof of 4.4 Proposition:
Since there is $w' \notin S$ which dominates $w \in S$, there must also exist $w^\wedge \in S$ such that $w^\wedge \succ w'$ where $w^\wedge \nsucc w$ and $w^\wedge \nprec w$ to preserve external stability. From 5.5, 5.6 Lemmas, $w^\wedge$ must appear in one of the following forms:

(a, w_2, w_2, b) where $a < \frac{w_1+w_4}{2}$ or else $w \succ w^\wedge$ if $\frac{w_1+w_4}{2} < \hat{w_1} < w_1$ and $w^\wedge \succ w$ if $\hat{w_1} > w_1$.
(c, w_2, d, w_4) where $c < \frac{w_1+w_3}{2}$ or else $w \succ w^\wedge$ if $\frac{w_1+w_3}{2} < \hat{w_1} < w_1$ and $w^\wedge \succ w$ if $\hat{w_1} > w_1$. ■

The following result comes from combining the previous two propositions with internal stability.

4.7 Proposition: In any stable set S, $w^* = \left(w_1, w_2, \left[0, \frac{w_3+w_4}{2}\right), \left(\frac{w_3+w_4}{2}, w_3 + w_4\right]\right) \notin S$.

Proof of 4.7 Proposition:
Suppose by way of contradiction that $w^* = \left(w_1, w_2, \left[0, \frac{w_3+w_4}{2}\right), \left(\frac{w_3+w_4}{2}, w_3 + w_4\right]\right) \in S$. By 4.4 Proposition, there is at least one $w^\wedge \in S$ where:

$w^\wedge = \left(\left[0, \frac{w_1+w_3}{2}\right), w_2, \left[w_1 + w_3, \frac{w_1+w_3}{2}\right), w_4\right)$ or
$w^\wedge = \left(\left[0, \frac{w_1+w_4}{2}\right), w_2, w_3, \left[w_1 + w_4, \frac{w_1+w_4}{2}\right)\right)$

Since $w_1^* \neq \hat{w_1}$, $w_3^* \neq \hat{w_3}$, $w_4^* \neq \hat{w_4}$, $w_2^* = \hat{w_2}$, then by 3.10 Proposition either $w' \succ w$ or $w^* \succ w'$. This contradicts the internal stability of S. ■

Due to the difficulties of finding stable sets, the weaker solution concept of self-protection is used as a stepping stone. A self-protected set is one which necessarily dominates only those allocations which dominate members of the set. Formally,

4.8 Definition: A set of allocations S is *self-protected* if S is internally stable and

(SP) (self-protection) for each $w \in S$ and any $w' \succ w$, there is some $w'' \in S$ such that $w'' \succ w'$.

Note that every stable set is self-protected, but a self-protected set need not satisfy external stability. However, it is much more straightforward to show that symmetric stable sets do not exist for the 4-person majority pillage game.

4.9 Proposition: No non-empty symmetric self-protected set exist for n=4 players. (Alternate proof of Jordan and Obadia, 2007.) 4.10 and 4.11 Lemmas were proven by Jordan and Obadia (2007). Their proofs are not included in this text.

4.10 Lemma: Let S be a symmetric internally stable set. Then for each $w \in S$, $\#\{w_i | i \in I\} \leq 2$.

4.11 Lemma: Let S be a symmetric self-protected set. Then for each $w \in S$, $\#\{i | w_i \geq \frac{1}{n}\} \geq \frac{n}{2}$

Proof of 4.9 Proposition: Suppose by way of contradiction that S is a symmetric self-protected set. By Chapter 3, we know S contains allocation w where $w_i \geq ½$. Without loss of generality, assume w is in standard form.

Case 1: Let $w_2 = w_3 = w_4$

Since $w_1 \geq ½$, $w_2 = w_3 = w_4 < ¼$. By 5.11 Lemma, $\#\{i | w_i \geq \frac{1}{n}\} \geq \frac{n}{2}$ in a symmetric self-protected set. In this case, $\#\{i | w_i \geq \frac{1}{n}\} = 1 \leq 2 = \frac{n}{2}$ which contradicts 5.11 Lemma.

Case 2: Let $w_2 \neq w_3$ or $w_3 \neq w_4$

Without loss of generality, consider the case where $w_2 \neq w_3$. Since $w_1 \geq ½$, $½ > w_2 \neq w_3$. Therefore, $\#\{w_i | i \in I\} \geq 3$ which contradicts 5.10 Lemma. ■

Since there are no non-empty symmetric self-protected sets for the 4-person majority pillage game, and any externally stable set is self-protected, no symmetric stable sets exist for this game, either.

5. Modeling a Majority Pillage Game

In an attempt to discover the maximum size of an internally stable set and to determine whether stable sets exist in the n=4 person majority pillage game, a java program was written to help accomplish these goals[8]. The program is broken into two larger methods, one of which is dedicated toward internal stability, and the other toward self-protection (and by consequence, external stability).

5.1 How the Program Works:

The program starts by generating a random 4-player wealth allocation around which it attempts to form an internally stable set. It then continues to generate allocations, and if each additional allocation satisfies the requirements for internal stability[9], it will add them to the set. If internal stability is not satisfied, then the program will reject the allocation and start again. After a pre-programmed number of consecutive failures, the program will end this method.

The second method begins in much the same way as the first. The program generates a random candidate wealth allocation, but this time tests for external stability. If there is no domination between those allocations within the set and the new candidate allocation, then the program will add this allocation to the set. If the candidate allocation is dominated by an allocation already in the set, the computer will generate a new allocation and try again. The program ends when every possible wealth allocations (for a prescribed number of decimal places) have been tested and the set is still found to be stable, or an allocation is found which demonstrates that the set is not externally stable (i.e. the allocation dominates at least one element in the internally stable set). If the former occurs, the program will return the results of the simulation as "STABLE" and if not, it will display the allocation which blocks the stability of the set.

5.2 Results:

The results of millions of simulations all indicate the same conclusion – there are no stable sets in the 4-person majority pillage game. This is strong evidence that such sets do not exist, which in and of itself furthers economists' knowledge of the majority pillage game. However, the upper bound on the size of the internally stable sets is also of note to the theory of pillage games. The largest sized set found using the program is a set containing 26 wealth allocations. However, this set is not unique. In fact, several internally stable sets of size 26 have been found. Whether there is a certain significance to the number 26, or if it is merely a coincidence has yet to be shown. As a point of illustration, two internally stable sets of size 26 have been reproduced below. Each row represents one allocation in the set, with each of the four numbers representing the proportion of wealth each player controls. The allocation which blocks external stability for each set (which is shown sandwiched between two rows of asterisks) is also included for each of these sets.

[8] The text of the computer program can be found in the Appendix. The program code includes significant in-text comments which should allow the interested reader to follow the program in a step-by-step fashion.

[9] Formal requirements for internal stability and self-protection can be found in Chapter 4.

Number of Successes: 26
0.596 0.266 0.100 0.038
0.013 0.004 0.708 0.275
0.360 0.502 0.100 0.038
0.360 0.502 0.068 0.070
0.005 0.012 0.708 0.275
0.005 0.012 0.398 0.585
0.596 0.266 0.039 0.099
0.012 0.005 0.398 0.585
0.458 0.329 0.194 0.019
0.393 0.394 0.194 0.019
0.458 0.329 0.030 0.183
0.000 0.038 0.489 0.473
0.038 0.000 0.489 0.473
0.003 0.030 0.497 0.470
0.377 0.410 0.030 0.183
0.038 0.000 0.474 0.488
0.000 0.038 0.462 0.500
0.030 0.003 0.497 0.470
0.288 0.212 0.251 0.249
0.288 0.212 0.244 0.256
0.249 0.251 0.244 0.256
0.249 0.251 0.298 0.202
0.285 0.215 0.298 0.202
0.030 0.003 0.470 0.497
0.240 0.260 0.251 0.249
0.240 0.260 0.246 0.254

****
0.343 0.519 0.068 0.070
****

Number of Successes: 26
0.350 0.364 0.164 0.122
0.040 0.128 0.309 0.523
0.133 0.035 0.309 0.523
0.473 0.325 0.032 0.170
0.330 0.442 0.010 0.218
0.288 0.510 0.032 0.170
0.428 0.344 0.010 0.218
0.008 0.203 0.451 0.338

0.040 0.128 0.515 0.317
0.103 0.065 0.515 0.317
0.209 0.002 0.451 0.338
0.008 0.203 0.368 0.421
0.209 0.002 0.374 0.415
0.024 0.170 0.330 0.476
0.368 0.346 0.164 0.122
0.309 0.486 0.014 0.191
0.257 0.243 0.279 0.221
0.249 0.251 0.279 0.221
0.249 0.251 0.208 0.292
0.270 0.230 0.208 0.292
0.270 0.230 0.256 0.244
0.231 0.269 0.256 0.244
0.231 0.269 0.237 0.263
0.170 0.024 0.330 0.476
0.024 0.170 0.476 0.330
0.170 0.024 0.476 0.330

****
0.458 0.486 0.050 0.006
****

5.3 Previous results:

This computer program is the first effort to find a limit on the size of internally stable sets in the majority pillage game (to the knowledge of this author). However, upper bounds for the size of internally stable sets have been found for pillage games in general. Specifically, Kerber and Rowat (2009) discuss a Ramsey bound on the size of an internally stable set. Unfortunately, this bound is difficult to calculate. Saxton (2009) generates a tighter, easier to calculate numerical upper bound on the size of such sets. The following theorem is reproduced from Saxton (2009, p 14):

"**Theorem 18**. *Let $S \subset A$ be a stable set for n players. Then*

$$|S| \leq 3^{2^n} - 1$$

In fact with a little more work one can tighten this bound to $3^{2^n - n - 2} \cdot 2^n + n$."

According to Saxton's findings, an internally stable set for the $n = 4$ majority pillage game can contain as many as $3^{16-4-2} \cdot 2^4 + 4$, or 944788, allocations. Clearly, this conflicts significantly with the maximum set of 26 elements found in the computer program. The most logical conclusion from this difference is that the majority pillage game has a significantly lower upper bound on the size of internally stable sets than pillage games in general.

6. Conclusions and Future Work

The implications of the majority pillage game are important. The framework can be used to model any situation in which agents fight over an item which makes them stronger. Examples could include nations fighting over natural resources, cavemen fighting over meat, or companies fighting over market share. The existence of a solution concept for this game for odd numbers of players is paramount in understanding cooperation and conflict in this model.

The analysis presented above indicates that conflict may be an unavoidable feature in anarchic situations. Since results suggest that stable sets do not exist, without a new solution concept, there may be no way to end the act of pillage. Nevertheless, this result is not known for sure. This thesis represents one step along the path of answering the questions of existence and uniqueness of stable sets in pillage games. Future work could include tightening the bound on the number of allocations possible in an internally stable set, and finding the minimum number of allocations which can exist in externally stable sets. Eventually, the goal is to be able to generalize these findings to even numbers of players.

Works Cited

Bott, R. "Symmetric solutions to majority games". Contributions of the Theory of Games, Vol. II, Kuhn, H.W. and A.W. Tucker (Eds.), Annals of Mathematical Studies No. 28 (1953) 3-27.

Chwe, M. "Farsighted coalitional stability". Journal of Economic Theory 63 (1994) 299-325.

Ghiselin, M. "The Economy of the Body". *The American Economic Review*, Vol. 68, No. 2, Papers and Proceedings of the Ninetieth Annual Meeting of the American Economic Association (May, 1978), pp. 233-237.

Gillies, D. "Solutions to general non-zero-sum games." Annals of Mathematical Studies No. 40, Princeton Univ. Press, Princeton, N. J., 1959, pp. 47-85.

Hirshleifer, J. "Anarchy and its breakdown". Journal of Political Economy 103 (1995) 26-52.

Jordan, J. "Pillage and property". Journal of Economic Theory 131 (2006) 26-44.

Jordan, J. and Obadia, D. "Stable sets in majority pillage games." 2007, unpublished paper.

Kerber, M. and Rowat, C. "A Ramsey Bound on Stable Sets in Jordan Pillage Games". 1 May, 2009. http://ssrn.com/abstract=1359328

Lucas, W. "On solutions for *n*-person games." RM-5567-PR, The RAND corp., Santa Monica, 1968.

Lucas, W. "Some recent developments in *n*-person game theory". SIAM Review 13 (1971) 491-523.

Lucas, W. "Von Neumann-Morgenstern stable sets". In R. Aumann, S. Hart (Eds.), Handbook of Game Theory, Elsevier, Amsterdam, 1992.

Lucas, W. and Michaelis, K. "Finite solution theory for coalitional games." SIAM Journal on Algebraic and Discrete Methods, 3 (1982) 551-565.

Lucas, W. et al. "A New Family of Finite Solutions." International Journal of Game Theory, Vol. 11, Issue 3/4 (1982) 117-127.

Luce, R. and Raiffa, H. Games and Decisions. Wiley, London, 1957.

Saxton, D. "Strictly monotonic multidimensional sequences and stable sets in pillage games." 2009, preprint.

Skaperdas, S. "Cooperation, conflict and power in the absence of property rights". American Economic Review 82 (1992) 720-739.

Von Neumann, J. and Morgenstern, O. Theory of Games and Economic Behavior. Wiley, New York, 1944.

Appendix – Computer Program

User Interface:

```
/**
 * Runs a user specified number of simulations
 *
 * @author
 * @version
 */
import java.util.*;
public class Driver
{
   public static void main(String [] args)
   {
      Random gen = new Random(); //Random Number generator
      final int numSimulations = 1; //Number of simulations
      final int numPlayers = 4; //Number of players
      final int numFailures = 100000; //Number of consecutive failures before an iteration ends
      final int numIterations = 100000; //Number of iterations in a single simulation
      final int numDecimals = 3; //Number of Decimals in allocation
      final int stabilityFailures = 100000; //Number of consecutive failures while checking external
stability
      //String pathName = "******************"; //Location where the results will be stored
      String pathName = "********************"; //Location where the results will be stored

      Simulation s = new Simulation(gen, numPlayers, numFailures, numIterations, numDecimals,
stabilityFailures, pathName);
      for(int x=0; x< numSimulations; x++)
      {
         s.run(pathName);
      }
      System.out.println("********************************");
      System.out.println("Max Set Size: " + s.maxSuccesses);
      System.out.println("Number of Stable Sets: " + s.numStableSets +'\n');
      System.out.println("********************************");
   }
}
```

Simulation:

```
/**
 * A simulation is a bunch of iterations. An iteration consists of a chain of allocations that ends
```

```
 * when a predetermined number of consecutive failures is encountered. Once an iteration produces a
set
 * of allocations, the set is tested to see if it is externally stable. Methods of testing include using
 * user-defined allocations, a list of randomly generated allocations for simulations with 3 or more
decimal
 * places, and a list of all possible allocations for simulations with less than 3 decimal places.
 *
 * @author
 * @version
 */
import java.util.*;
import java.io.*;
import java.text.*;
public class Simulation
{
   //PIVs
   Random gen;              //Random number generator
   int numPlayers;          //Number of players in the simulation
   int numFailures;         //Number of consecutive failures before an iteration ends
   int numIterations;        //Number of iterations in a single simulation
   int numDecimals;          //Number of decimal places in the allocations
   int successes;           //Number of successful allocations in an iteration
   int maxSuccesses;         //Number of successes in longest chain
   int stabilityFailures;    //Number of consecutive failures when determining external stability
   int numStableSets;        //Number of stable sets found during simulation
   ArrayList allocations;    //Array of the current successful allocations
   ArrayList userInternal;    //Array Containing user-defined allocations for inclusion within the set
   ArrayList userExternal;    //Array Containing user-defined allocations for external stability testing
   String path;             //Path where input and output files are held

/**This sets up the simulation before running it----------------------------------------------------------------*/
   //Constructor
   public Simulation(Random g, int np, int nf, int ni, int nd, int sf, String p)
   {
      gen = g;
      numPlayers = np;
      numFailures = nf;
      numIterations = ni;
      numDecimals = nd;
      stabilityFailures = sf;
      successes = 0;
      numStableSets = 0;
      maxSuccesses = 0;
      allocations = new ArrayList();
      path = p;
      userInternal = new ArrayList();
      userExternal = new ArrayList();
   }
```

```
/**This section of code governs the way the program runs at each level (simulation, iteration, round)----------*/

   //This method runs one series of iterations known as a simulation. The results are printed to a specified text file
   public void run(String path)
   {
      // Storage Arrays and incrementor
      ArrayList c = new ArrayList();
      double[] z = new double[4];
      int count = 0;

      //List of user-defined allocations for inclusion within the set
      try
      {
         Scanner con1 = new Scanner(new File(path+"/internal.txt"));
         while(con1.hasNext()) //Looks at every element in the file
         {
            count++;
            //Stores four wealths to one allocation
            for(int q=0; q<4; q++)
            {
               z[q] = con1.nextDouble()*Math.pow(10, numDecimals);
            }
            if(validAllocation(z)) //Rejects allocation if it is invalid (wealths do not sum to 1, improper decimal places)
               userInternal.add(z);
            else
               System.out.println("Allocation number " + count + " in internal is not a valid allocation");
         }
      }
      catch(Exception e)
      {
         System.out.println("Internal file not found");
      }
      //List of user-defined allocations for external stability testing
      count = 0;
      try
      {
         Scanner con2 = new Scanner(new File(path+"/external.txt"));
         while(con2.hasNext()) //Looks at every element in the file
         {
            count++;
            //Stores four wealths to one allocation
            for(int r=0; r<4; r++)
            {
               z[r] = con2.nextDouble()*Math.pow(10, numDecimals);
```

```
            }
            if(validAllocation(z)) //Rejects allocation if it is invalid (wealths do not sum to 1, improper
decimal places)
               userExternal.add(z);
            else
               System.out.println("Allocation number " + count + " in external is not a valid allocation");
         }
      }
      catch(Exception e)
      {
         System.out.println("External file not found");
      }
      for(int j=0; j< numIterations; j++)
      {
         runIteration();
         z = stability();
         if(z == null)
            numStableSets++;
         writeOutput();
         writeStabilityOutput(z);
         if(successes >= 16)
         {
            writeMaxOutput();
            writeMaxStabilityOutput(z);
         }
         if(successes > maxSuccesses)
         {
            maxSuccesses = successes;
         }
         successes = 0;
         allocations.clear();
         z = null;
      }
      writeFinalOutput();
   }

   //This method runs an iteration (a series of allocations that continues until a predetermined amount
of failures is encountered)
   public void runIteration()
   {
      int round = 1; //Round number (used for determining allocation assignment)
      int consFailures = 0; //Number of current consecutive failures
      double[] roundAllocations = new double[numPlayers]; //List of allocations for a single round

      //Add in user defined allocations
      for(int i=0; i<userInternal.size(); i++)
      {
         //Determine validity and add
```

```
        if(determineValidity((double[])userInternal.get(i), round))
        {
          successes++; //Increment number of successful rounds in this allocation
          allocations.add(roundAllocations);
        }
        round++;
      }
      //Iteration continues rounds until a predetermined amount of consecutive failures is encountered
      while(consFailures < numFailures)
      {
        roundAllocations = runRound(round); //set values of allocations for the current round
        //Determine validity of round allocations
        if(determineValidity(roundAllocations, round))
        {
          successes++; //Increment number of successful rounds in this allocation
          consFailures = 0; //Consecutive Failures ends so reset to 0
          allocations.add(roundAllocations);
        }
        else
        {
          consFailures++; //Increment the number of consecutive failures
        }
        round++; //Increment round number
      }
    }

    //This method determines which algorithm to use to assign allocations
    public double[] runRound(int n)
    {
      //Odd round number
      if((n%2)!=0)
      {
        return oddRound();
      }
      //Special designation used for external stability checks
      else if(n == 0)
      {
        return otherRound();
      }
      //Even round number
      else
      {
        return evenRound();
      }
    }

/**This section of code generates the allocations for inclusion or exclusion within the set----------
----------*/
```

```
   //Determines whether an allocation is valid (elements sum to 1 and all elements have the proper
number of decimals)
   //PARAMETERS:
   //        a - possible allocation
   //RETURN:
   //        True - the allocation is valid
   //        False - the allocation is invalid
   public boolean validAllocation(double[] a)
   {
      double sum = 0; //Sum of wealths for four players
      int diff = 0;   //Number of players with an invalid number of decimals

      //Sums the elements of the allocation
      for(int j=0; j<4; j++)
      {
         sum += a[j];
         if((a[j]%1)!=0)
            diff++;
      }
      //identifies if allocation is valid and returns result
      if((sum == Math.pow(10, numDecimals))&&(diff==0))
         return true;
      return false;
   }

   //This method assigns allocations for an odd round
   //ALGORITHM:    The order of the players is randomized, then the first three selected players receive
random wealths
   //            and the remaining player receives an amount such that all wealths sum to 1
   public double[] oddRound()
   {
      final double total = Math.pow(10,numDecimals);  //Remaining amount of allocations
      double sum = 0;                          //Current sum of alocations
      double[] a = new double[numPlayers];          //Array containing allocations for current round
      int index = 0;                           //Index of current player
      ArrayList ids = new ArrayList();             //ArrayList used to randomly select the order allocations are
given at

      //Randomize order in which allocations are given
      for(int i=0; i<4; i++)
      {
         ids.add(new Integer(i));
      }
      //Each player is assigned an allocation according to the algorithm (random allocations that sum to 1
in decimal form)
      for(int j=0; j<4; j++)
      {
```

```
            index = ((Integer)ids.remove(gen.nextInt(ids.size()))).intValue();
            //Sets the last allocation such that all allocations sum to total
            if(j == 3)
            {
               a[index] = total - sum;
               sum += a[index];
            }
            //Randomly sets an allocation based on the difference between total and sum
            else
            {
               a[index] = (double)gen.nextInt((int)(total-sum+1));
               sum += a[index];
            }
         }
         return a;
      }

      //This method assigns allocations for an even round
      //ALGORITHM:    The order of the players is randomized, and one of the allocations already in the list
is randomly
      //            selected. Two of the players are given wealths equal to the ones from the previous
allocation, the third player
      //            receives a random allocation, and the remaining player receives a wealth such that all
players wealths sum to 1
      public double[] evenRound()
      {
         final double total = Math.pow(10,numDecimals); //Remaining amount of allocations
         double sum = 0; //Sum of allocations
         double[] a = new double[numPlayers]; //Array containing allocations for current round
         double[] b = (double[])allocations.get(gen.nextInt(allocations.size())); //Allocations from a previous
success
         int index = 0; //index of current player
         ArrayList ids = new ArrayList(); //ArrayList used to randomly select the order allocations are given at

         //Randomize order in which allocations are given
         for(int i=0; i<4; i++)
         {
            ids.add(new Integer(i));
         }

         //Assign allocations
         for(int j=0; j<4; j++)
         {
            //Random index
            index = ((Integer)ids.remove(gen.nextInt(ids.size()))).intValue();
            //Assigns two players to have same allocations as last successful allocation
            if(j<2)
            {
```

```
            a[index] = b[index];
            sum += a[index];
         }
         //The last random player receives what is left over
         else if(j==3)
         {
            a[index] = total - sum;
            sum += a[index];
         }
         //The third random player receives a random allocation based on what is left over
         else
         {
            a[index] = (double)gen.nextInt((int)(total-sum+1));
            sum += a[index];
         }
      }
      return a;
   }

   public double[] otherRound()
   {
      final double total = Math.pow(10,numDecimals); //Remaining amount of allocations
      double sum = 0; //Sum of allocations
      double[] a = new double[numPlayers]; //Array containing allocations for current round
      double[] b = (double[])allocations.get(gen.nextInt(allocations.size())); //Allocations from a previous
success
      int index = 0; //index of current player
      ArrayList ids = new ArrayList(); //ArrayList used to randomly select the order allocations are given at

      //Randomize order in which allocations are given
      for(int i=0; i<4; i++)
      {
         ids.add(new Integer(i));
      }

      //Assign allocations
      for(int j=0; j<4; j++)
      {
         //Random index
         index = ((Integer)ids.remove(gen.nextInt(ids.size()))).intValue();
         //Assigns one player to have same allocations as random successful allocation
         if(j<1)
         {
            a[index] = b[index];
            sum += a[index];
         }
         //The last random player receives what is left over
         else if(j==3)
```

```
        {
           a[index] = total - sum;
           sum += a[index];
        }
        //Two players are given random wealths
        else
        {
           a[index] = (double)gen.nextInt((int)(total-sum+1));
           sum += a[index];
        }
     }
     return a;
  }

/**This section of code determines whether a potential allocation belongs in the set----------
----------*/

  //This method determines whether the current rounds allocation is valid relative to all past successes
  public boolean determineValidity(double[] a, int n)
  {
     double[] b = new double[numPlayers]; //Array that holds values of past successful allocations
     //First allocation is always successful
     if(n==1)
     {
        return true;
     }
     //Determine if allocation is valid relative to all past successes
     for(int x=0; x<allocations.size(); x++)
     {
        b = (double[])allocations.get(x);
        //If allocation fails relative to even one past successes it is not a valid allocation
        if(!(optionOne(a, b) || optionTwo(a, b)))
        {
          return false;
        }
     }
     //If it reaches this point it is a valid allocation
     return true;
  }

  //This method describes one way in which an allocation may be valid
  public boolean optionOne(double[] a, double[] b)
  {
     int better = 0; //Number of players that are better off
     final int betterRequired = 2; //Number of players required to be better off
     double sumBetterA = 0; //Sum of the current allocations from players who are better off in current
round
```

```
    double sumBetterB = 0; //Sum of the previous allocations from players who are better off in current
round

    //Compare values between rounds
    for(int i=0; i<a.length; i++)
    {
      //fails if any values are equal between rounds
      if(a[i] == b[i])
      {
        return false;
      }
      //evaluation of players that are better off
      else if(a[i]>b[i])
      {
        better++;
        if(better>betterRequired)
        {
          return false;
        }
        sumBetterA += a[i];
        sumBetterB += b[i];
      }
    }
    // fails if only 1 player is better off
    if(better < betterRequired)
    {
      return false;
    }
    //The two players that are better off must have greater than half of the allocations in the current
round
    //and less than half of the allocations in the past allocation
    if((sumBetterA >= (.5*Math.pow(10, numDecimals))) && (sumBetterB <= (.5*Math.pow(10,
numDecimals))))
    {
      return true;
    }
    //If it reaches here the allocation is not valid according to this allocation
    return false;

  }

  //This method describes a second way in which the current rounds allocation may be successful
  public boolean optionTwo(double[] a, double[] b)
  {
    int same = 0; //Number of players that have the same allocation between rounds
    final int sameRequired = 2; //Number of players required be the same between allocations
    int betterIndex = 0; //Index of better off player
    int worseIndex = 0; //Index of worse off player
```

```
      //Compare values between rounds
      for(int i=0; i<a.length; i++)
      {
         //fails if more than two are the same between rounds
         if(a[i] == b[i])
         {
            same++; //Icrement the number of players that have the same allocation between rounds
            if(same > sameRequired)
            {
               return false;
            }
         }
         //evaluation of players that are better off
         else if(a[i]>b[i])
         {
            betterIndex = i;
         }
         else
         {
            worseIndex = i;
         }
      }
      //fails if only 1 player is the same
      if(same < sameRequired)
      {
         return false;
      }
      //The better off player must have a larger allocation in the current round than the worse off player
in the current round.
      //Also the better off player must have a smaller allocation in the past round than the worse off
player in the past round.
      if((a[betterIndex]>a[worseIndex])&&(b[betterIndex]<b[worseIndex]))
      {
         return true;
      }
      //If it reaches here the allocation is not valid according to this algorithm
      return false;
   }

/**This section of code is devoted to determining the external stability of the set----------------------------
--*/

   //Attempts to determine if set is externally stable and returns any allocations that dominate the set
   //RETURN:
   //        An allocation that proves the set is not externally stable (null if externally stable)
   public double[] stability()
   {
```

```
    int failures = 0;          //Number of consecutive failures
    int rand = 0;              //Number of randomly selected round (0, 1, 2)
    double[] a = new double[4]; //Storage for results

    //Checks external stability with user defined allocations
    a = userStabilityCheck();
    if(a != null)
      return a;
    //Checks external stability with randomly generated allocations
    while((failures < stabilityFailures))
    {
      //Set allocation (randomly chooses algorithm)
      rand = gen.nextInt(3);
      a = runRound(rand);
      if(determineValidity(a, 0))
      {
        allocations.add(a);
        successes++;
        failures = 0;
      }
      //Checks whether the set is externally stable to the random allocation a
      else
      {
        if(!checkStability(a))
        {
          return a;
        }
        else
        {
          failures++;
        }
      }
    }
    //Checks stability versus every allocation as a last resort
    //totalStabilityCheck();
    return null;
  }

  //This method determines whether a set of allocations is externally stable relative to a new
allocation(a)
  //PARAMETERS:
  //        a - test allocation
  //RETURN:
  //        True - The set is externally stable with respect to a
  //        False - The set is dominated by a
  public boolean checkStability(double[] a)
  {
    double[] b = new double[4]; //allocation from set
```

```
    //Checks whether the new allocation dominates all elements of the set
    for(int i=0; i<allocations.size(); i++)
    {
       b = (double[])allocations.get(i); //allocation from set
       //indicates a member of the set dominated a
       if(checkDominance(b, a))
          return true; //The set is externally stable with respect to a
    }
    //The set is dominated by a
    return false;
  }

  //Checks whether an allocation a dominates an allocation b
  //PARAMETERS:   a - member of set
  //            b - new allocation
  //RETURN:   true - a dominates b
  //         false - a does not dominate b
  public boolean checkDominance(double[] a, double[] b)
  {
    int same = 0;   //Number of numbers that are the same between sets
    int index = -1; //Id of player investigated
    double max = 0; //Max allocation
    int worse = 0;  //Number of worse off players
    int index2 = 0; //Second stored id

    // Find number of players with same values between allocations
    for(int x=0; x<a.length; x++)
    {
       if(a[x] == b[x])
          same++;
    }
    //The sets are identical so a does not dominate b
    if((same == 4) || (same == 3))
    {
       return false;
    }
    //a does not dominate b if two players have the same wealth between allocations, and the richer of
the two remaining in allocation a is
    //worse off (the player's wealth in a is less than it is in b)
    else if(same == 2)
    {
       for(int i=0; i<4; i++)
       {
          if((a[i] != b[i]) && (a[i] > max))
          {
             index = i;
             max = a[i];
```

```
            }
        }
        if(a[index] <  b[index])
        {
            return false;
        }
    }
    //a does not dominate b if one player has an equal wealth in each allocation, and two players in allocation a are worse off, which
    //means their wealth in allocation a is less than their wealth in allocation b
    else if(same == 1)
    {
        for(int i=0; i<4; i++)
        {
            if(a[i] < b[i])
                worse++;
        }
        if(worse == 2)
            return false;
    }
    //a does not dominate b if when no player has an equal wealth between allocations either three players are worse off (their wealth in a
    //is less than their wealth in b), or two players are worse off and they have more than half of the wealth in allocation b
    else
    {
        for(int i=0; i<4; i++)
        {
            if(a[i] < b[i])
            {
                worse++;
                if(index < 0)
                    index = i;
                else
                    index2 = i;
            }
        }
        if(worse == 3)
        {
            return false;
        }
        else if(worse == 2)
        {
            if((b[index] + b[index2]) > (.5*Math.pow(10, numDecimals)))
                return false;
        }
    }
    //If the code reaches this point it indicates that a does dominate b
```

```
   return true;
}

//Checks the external stability of a set using user defined allocations
public double[] userStabilityCheck()
{
   double[] a = new double[4]; //user defined allocation

   //Determines whether any of the allocations dominate the set
   for(int j=0; j<userExternal.size(); j++)
   {
      a = (double[])userExternal.get(j);
      //If the allocation belongs in the set it is added
      if(determineValidity(a, 0))
      {
         allocations.add(a);
         successes++;
      }
   }
   for(int k=0; k<userExternal.size(); k++)
   {
      if(!checkStability(a))
         return a;
   }
   return null;
}

//Checks the external stability of a set using user defined allocations
public double[] totalStabilityCheck()
{
   double[] a = new double[4]; //storage for allocation

   for(int i=0; i<=Math.pow(10, numDecimals); i++)
   {
      for(int j=0; j<=Math.pow(10, numDecimals)-i; j++)
      {
         for(int k=0; k<=Math.pow(10, numDecimals)-i-j; k++)
         {
            a[0] = i;
            a[1] = j;
            a[2] = k;
            a[3] = Math.pow(10, numDecimals)-i-j-k;
            //If the allocation belongs in the set it is added
            if(determineValidity(a, 0))
            {
               allocations.add(a);
               successes++;
            }
```

```
            else if(!checkStability(a))
            {
               return a;
            }
         }
      }
   }
   return null;
}

/**This section code is devoted entirely to methods that create output----------
----------*/

  //The results for each iteration are printed to a text file that is named based on the parameters of the
simulation
  //     The number of successes in each iteration along with the actual successful allocations are writen
to a text file
  //     FORMAT: Each row is a successful allocation
  //             Each column is a specific player
  //             The dashed lines separate iterations
  public void writeOutput()
  {
     try
     {
        //Open file (appends - all data is ended to end of file in case the file already exists)
        BufferedWriter writer = new BufferedWriter(new
FileWriter(path+"/simP"+numPlayers+"F"+numFailures+"D"+numDecimals+".txt",true)) ;
        DecimalFormat decimal = new DecimalFormat();

        //Formatting
        decimal.setMaximumFractionDigits(numDecimals);
        decimal.setMinimumFractionDigits(numDecimals);

        //Print results for the iteration
        writer.write("-------------------------------------------------------------------------------" + '\n');
        writer.write("Number of Successes: " + successes+'\n');
        double[] a = new double[numPlayers];
        for(int x=0; x< allocations.size(); x++)
        {
           a = (double[])allocations.get(x); //Allocations from successful round
           for(int y=0; y<a.length; y++)
           {
              //Forms a row of a successful round's allocations
              writer.write(decimal.format(a[y]/Math.pow(10, numDecimals)) + "  ");
           }
           writer.write('\n');
        }
        writer.write("-------------------------------------------------------------------------------"+ '\n');
```

```
            writer.write('\n');

            //Close File
            writer.close();
        }
        //In case file cannot be opened
        catch(Exception e)
        {
            System.out.println("Failed to open file");
        }
    }

    //Displays the length of the list in the text file
    public void writeFinalOutput()
    {
        try
        {
            //Open file (appends - all data is ended to end of file in case the file already exists)
            BufferedWriter writer = new BufferedWriter(new
FileWriter(path+"/simP"+numPlayers+"F"+numFailures+"D"+numDecimals+".txt",true)) ;
            DecimalFormat decimal = new DecimalFormat();

            //Formatting
            decimal.setMaximumFractionDigits(numDecimals);
            decimal.setMinimumFractionDigits(numDecimals);

            //Print results for the iteration

writer.write("**********************************************************************
****" + '\n');
            writer.write("Max Successes: " + maxSuccesses +'\n');
            writer.write("Number of Stable Sets: " + numStableSets +'\n');

writer.write("**********************************************************************
****"+ '\n');
            writer.write('\n');

            //Close File
            writer.close();
        }
        //In case file cannot be opened
        catch(Exception e)
        {
            System.out.println("Failed to open file");
        }
    }

    //Outputs wheter or not a set is externally stable to the output text file
```

```
   public void writeStabilityOutput(double[] a)
   {
      try
      {
         //Open file (appends - all data is ended to end of file in case the file already exists)
         BufferedWriter writer = new BufferedWriter(new
FileWriter(path+"/simP"+numPlayers+"F"+numFailures+"D"+numDecimals+".txt",true)) ;
         DecimalFormat decimal = new DecimalFormat();

         //Formatting
         decimal.setMaximumFractionDigits(numDecimals);
         decimal.setMinimumFractionDigits(numDecimals);

         //Print the dominant allocation if it exists, otherwise tell it is stable

writer.write("**********************************************************************
****" + '\n');
         if(a != null)
         {
            for(int y=0; y<4; y++)
            {
               //Forms a row of the dominant allocations terms
               writer.write(decimal.format(a[y]/Math.pow(10, numDecimals)) + "  ");
            }
            writer.write('\n');
         }
         else
         {
            writer.write("STABLE!!!"+'\n');
         }

writer.write("**********************************************************************
****"+ '\n');
         writer.write('\n');
         //Close File
         writer.close();
      }
      //In case file cannot be opened
      catch(Exception e)
      {
         System.out.println("Failed to open file");
      }
   }

   //Outputs the contents of large lists to a separate file for easier access
   public void writeMaxOutput()
   {
      try
```

```
    {
      //Open file (appends - all data is ended to end of file in case the file already exists)
      BufferedWriter writer = new BufferedWriter(new
FileWriter(path+"/MaxsimP"+numPlayers+"F"+numFailures+"D"+numDecimals+".txt",true)) ;
      DecimalFormat decimal = new DecimalFormat();

      //Formatting
      decimal.setMaximumFractionDigits(numDecimals);
      decimal.setMinimumFractionDigits(numDecimals);

      //Print results for the iteration
      writer.write("--------------------------------------------------------------------------------" + '\n');
      writer.write("Number of Successes: " + successes+'\n');
      double[] a = new double[numPlayers];
      for(int x=0; x< allocations.size(); x++)
      {
        a = (double[])allocations.get(x); //Allocations from successful round
        for(int y=0; y<a.length; y++)
        {
          //Forms a row of a successful round's allocations
          writer.write(decimal.format(a[y]/Math.pow(10, numDecimals)) + "  ");
        }
        writer.write('\n');
      }
      writer.write("--------------------------------------------------------------------------------"+ '\n');
      writer.write('\n');

      //Close File
      writer.close();
    }
    //In case file cannot be opened
    catch(Exception e)
    {
      System.out.println("Failed to open file, 1");
    }
  }

  //Displays whether a large set is stable or not
  public void writeMaxStabilityOutput(double[] a)
  {
    try
    {
      //Open file (appends - all data is ended to end of file in case the file already exists)
      BufferedWriter writer = new BufferedWriter(new
FileWriter(path+"/Max"+"simP"+numPlayers+"F"+numFailures+"D"+numDecimals+".txt",true)) ;
      DecimalFormat decimal = new DecimalFormat();

      //Formatting
```

```
        decimal.setMaximumFractionDigits(numDecimals);
        decimal.setMinimumFractionDigits(numDecimals);

        //Print the dominant allocation if it exists, otherwise tell it is stable

writer.write("**********************************************************************
****" + '\n');
        if(a != null)
        {
           for(int y=0; y<4; y++)
           {
              //Forms a row of the dominant allocations terms
              writer.write(decimal.format(a[y]/Math.pow(10, numDecimals)) + "  ");
           }
           writer.write('\n');
        }
        else
        {
           writer.write("STABLE!!!"+'\n');
        }

writer.write("**********************************************************************
****"+ '\n');
        writer.write('\n');
        //Close File
        writer.close();
     }
     //In case file cannot be opened
     catch(Exception e)
     {
        System.out.println("Failed to open file, 2");
     }
  }

  public void writeAllocation(double[] a)
  {
     try
     {
        //Open file (appends - all data is ended to end of file in case the file already exists)
        BufferedWriter writer = new BufferedWriter(new FileWriter(path+"/test.txt",true)) ;
        DecimalFormat decimal = new DecimalFormat();

        //Formatting
        decimal.setMaximumFractionDigits(numDecimals);
        decimal.setMinimumFractionDigits(numDecimals);

        //Print the wealths within the allocation
        for(int y=0; y<a.length; y++)
```

```
            {
               //Forms a row of a successful round's allocations
               writer.write(decimal.format(a[y]/Math.pow(10, numDecimals)) + "  ");
            }
            writer.write('\n');

            //Close File
            writer.close();
         }
         //In case file cannot be opened
         catch(Exception e)
         {
            System.out.println("Failed to open file");
         }
      }
   }
```

Academic Vita of Bobak Pakzad-Hurson

Address: 109 Ghaner Drive
State College, PA 16803

E-Mail Id: bobby.ph@gmail.com

Education

Pennsylvania State University (Graduating with Honors in May 2011)
B.S. in Economics, Minor in Mathematics
Thesis: Non-symmetric stable sets in the majority pillage game (Supervised by Dr. James Jordan)

Work Experience

Date: June – August 2010
Title: Summer Consultant
Description: Statistical and data analysis
Institution: Bates White, LLC

Date: January 2009 – September 2010
Title: Research Assistant
Institution: Penn State Department of Economics
Supervisor's Name: Dr. James Jordan

Date: September 2008 – December 2010
Title: Teaching Assistant
Institution: Penn State Department of Economics

Awards

- Phi Beta Kappa| 2010
- Linda Brodsky Strumpf Trustee Scholarship, College of Liberal Arts | 2010-11
- Evan Pugh Scholar Award | 2009-10
- Goldstein Honors Scholarship | 2009-10
- President Spark's Award | 2008-09
- Rein Scholarship, College of Liberal Arts | 2008-09
- Omicron Delta Epsilon | 2010
- Newman Most Outstanding Junior Award, Department of Economics |2009-10
- Donato Trustee Scholarship | 2009-10
- David Grow Economics Fellowship | 2009-10
- Lombra Scholarship, Department of Economics | Spring, 2009
- President's Freshman Award | 2007-08

Skills:

Programming:	C++, SQL
Statistical Software:	Stata
MS Office:	Word, PowerPoint, Excel
Languages:	English (native), Farsi (fluent), Spanish (basic)