Solving Bluff

Marc Lanctot and Jeff Long
Department of Computing Science, University of Alberta
Edmonton, Alberta, Canada T6G 2E8
{lanctot|jlong1}@cs.ualberta.ca

Abstract

Bluff is a popular imperfect information dice-game that can be played with 2 or more players. In this paper, we find an optimal game-theoretic solution for a small scale game of Bluff using the game-solving method of Koller, Megiddo and von Stengel. We find that the game of 1 die vs. 1 die Bluff is slightly biased in favor of the second player to move, but the addition of a common variant balances the game, giving an advantage to neither player. We also show the optimal strategies required by both players to achieve these optimal values.

Introduction

Bluff is a simple dice-game involving both chance and imperfect information. To find the *optimal minimax strategy* to such a game would traditionally involve converting the game from its *extensive*, or game tree, form to its *normal*, or matrix, form. Once converted to such a form, the game's optimal strategies and its game-theoretic *value* can be computed using any linear program solver.

The problem is that a direct conversion from extensive to normal form results in a matrix that is exponential in the size of the game tree. This makes it infeasible to solve a game like Bluff even on the smallest scale. In 1994, Koller, Megiddo and von Stengel presented a method for solving extensive form games that is linear in the game tree size rather than exponential (Koller, Megiddo, & von Stengel 1994).

The goal of this paper is to use the method by Koller, Megiddo and von Stengel to find game-theoretic solutions to a small-scale game of Bluff.

This paper is organized as follows. The remainder of this section presents the rules of Bluff, and details the exact parameters for which we will solve the game. The next section outlines the key aspects of the method by Koller, Megiddo and von Stengel. The following section describes our methodology in applying the technique of Koller, Megiddo and von Stengel to Bluff. The fourth section presents the solution to the game and a discussion of our results, and the last section concludes the paper.

The Game of Bluff

Bluff (also commonly known as Liar's Dice and Perudo) is an imperfect information game for n players. Each player in the game has m dice. In practice, these are always 6-sided dice, but from a theoretical perspective, these dice could have any number of sides, which we denote by d.

The game consists of several rounds, depending on both the number of players and dice involved. At the start of every round, each player rolls their dice and hides the result under a cup. Players examine the results of their own roll, but not the rolls of their competitors. For the remainder of the round, the players take alternating turns. When it is the turn of player X to play, X must do one of two things: make a bid, or call Bluff. A bid consists of a quantity, q, and a die-face, f, and by making such a bid, the player X is saying that over all the dice rolled by all players, at least q of them are showing face f. The highest die-face, d, is considered to be 'wild,' meaning it simultaneously counts as being every other die face for the purpose of verifying bids.

Unless player X is starting the round, there will always be previous bid, b_{i-1}, made by the player whose turn immediately preceded the turn of X. If X wishes to make a bid, that bid must be higher than b_{i-1}. Bids are ordered first according to q, and then according to f (eg. a bid of 3-1's is higher than a bid of 2-5's). Since a bid where $f = d$ is less probable than any other bid of the same q, bidding in $f = d$ falls outside this normal progression. In general, a bid (q, f) where $f = d$ is considered to be higher than a bid (q_2, f_2) where $f_2 \neq d$ for all $q_2 < 2 * q$.

If instead of making a bid, X calls Bluff, then the current round comes to an end. All players reveal their dice, and the bid b_{i-1} made by player Y immediately prior to X's turn is verified. If the bid was in fact correct (ie. there are at least q dice showing either face f or the wild face, d), X loses the round (which usually means forfeiting a die for the next round). However, if the bid was not correct (Y was bluffing), then Y loses the round. When a player loses all their dice, they lose and are out of the game.

In this paper, our goal is to completely solve, in the game-theoretic sense, the game of 2-player Bluff where each player has one 6-sided die ($n = 2$, $m = 1$, $d = 6$).

There are numerous possible variants to Bluff. One variant that we will examine in this paper is 'checking.' Checking means that on player X's turn, in addition to making a bid or calling Bluff, they make opt to call Check instead. If a player calls Check, all dice are revealed, just as when Bluff is called, except that in this case, the checking player wins if

		Player 2	
		a	b
Player 1	a	−2,−2	−10,0
	b	0,−10	−5,−5

Figure 1: The payoff matrix for the Prisoner's Dilemma

the previous bid b_{i-1} was *exactly* correct (ie. there are precisely q dice showing either face f or the wild face, d). In all other cases, the checking player loses. In our experimental results, we will show how the addition of the option to check changes the game.

Background

This section presents the foundation (theory and algorithms) on which the the method described in the next section is built. The method used to find equilibrium strategies in Bluff is largely based on the ones presented by Daphne Koller et. al.. We suggest referring to (Koller, Megiddo, & von Stengel 1994) for a more elaborate and detailed version of this background information.

Classic Two-Person Game Theory

A game is said to be represented in *normal form* if it is represented by payoff matrices for player 1 and 2. Games that are represented in normal form are called *normal form games*, or sometimes also called *matrix games*. Hereon, we assume that a game is never played by more or less than two players. The m rows of a payoff matrix correspond to the actions that player 1 can take and the n columns of of a payoff matrix correspond to the actions that player 2 can take. The playout of a normal form game is a pair of choices ("strategies") made by player 1 and player 2, each made uninformed of the other's decision ("simultaneously"). The entries of the payoff matrices represent a payoff to the player; by convention, the matrix $\mathbf{A}$ is the payoff matrix for player 1 and $\mathbf{B}$ is the payoff matrix for player 2. Players are usually interested in maximizing this payoff. A common example of a normal form game is the *Prisoner's Dilemma*, depicted below in Figure 1. Note that the two matrices are typically shown overlaid: each entry a_{ij}, b_{ij} represents both the the payoff for player 1 if she plays strategy i while player 2 plays strategy j and the payoff for player 2 if she plays strategy j while player 1 plays strategy i.

Zero-sum games are special cases of normal form games where $\mathbf{B} = -\mathbf{A}$. These games are special in the sense that for every game played, either player 1's gain is player 2's loss or vice versa. They are also usually represented by a single payoff matrix (by convention, $\mathbf{A}$). Another common example is *Rock, Paper, Scissors* shown in Figure 2.

A *pure strategy*, sometimes also called a *deterministic strategy*, is a strategy where the player always takes the same action. A general or *mixed* strategy is a probability distribution over pure strategies. By convention, $\mathbf{x}$ is a strategy for player 1 and $\mathbf{y}$ is a strategy for player 2. If these strategies

		P2 = y		
		rock	paper	scissors
P1 = x	rock	0	−1	+1
	paper	+1	0	−1
	scissors	−1	+1	0

payoff matrix = A

Figure 2: The payoff matrix for player 1 for the game of Rock, Paper, Scissors

are used, it is easy to verify that the *expected payoff* to player 1 will be $\mathbf{x}^T\mathbf{A}\mathbf{y}$. A major result of the Minimax Theorem (von Neumann 1928) is that every two-person, zero-sum game has at least one pair of strategies that maximize their respective payoffs. These pairs of strategies are called *equilibrium strategies*. Note that Nash later proved that equilibrium strategies exist in all general-sum games (Nash 1950).

The usual way of finding equilibrium strategies in a two-person games is to formalize the problem as a linear program and use existing methods to solve such linear programs. The constraints follow from the definition of two-person zero-sum games: $\sum_{x_i \in \mathbf{x}} x_i = 1$ because the strategy is a probability distribution, and $\mathbf{x} \geq \mathbf{0}$ because the entries represent probabilities. If player 1 plays with a fixed strategy, then a best response strategy for player 2 can be found by finding the solution to the following linear program

$$\text{maximize}_{\mathbf{y}}(\mathbf{x}^T\mathbf{B}\mathbf{y}) \text{ subject to } \mathbf{F}\mathbf{y} = \mathbf{f}, \mathbf{y} \geq \mathbf{0}$$

where $\mathbf{F}$ is a matrix of 1's and $\mathbf{f}$ = 1. An analogous program can be derived for player 1 assuming player 2 employs a fixed strategy. In fact, using strong duality of linear programming the duals of these programs can be combined with the original programs to form 2 linear programs whose solutions correspond to equilibrium strategies for both players.

Games can also be expressed in *extensive form*, ie. by a *game tree*: a set of states and transitions corresponding to individual actions, where leaves are the ending positions, and any path from root to leaf in the tree corresponds to a particular playout. One way to find equilibrium strategies is to first convert the game to normal form and then to use linear programming. This method will work, but computational constraints make it impractical since the size of the matrix obtained is exponential in the size of the game tree itself. An example of an extensive form game is shown in Figure 3. Its normal form is given in Table 1 [1].

Imperfect Information Games

In an imperfect information game, such as Bluff, states which players cannot differentiate are called *information sets*. For example, in Figure 3 there are three information sets for player 1 and 2 information sets for player 2. An information set for a player is what that players knows about

[1]Note that there are errors in the values given in original paper; these have been corrected in the table shown here.

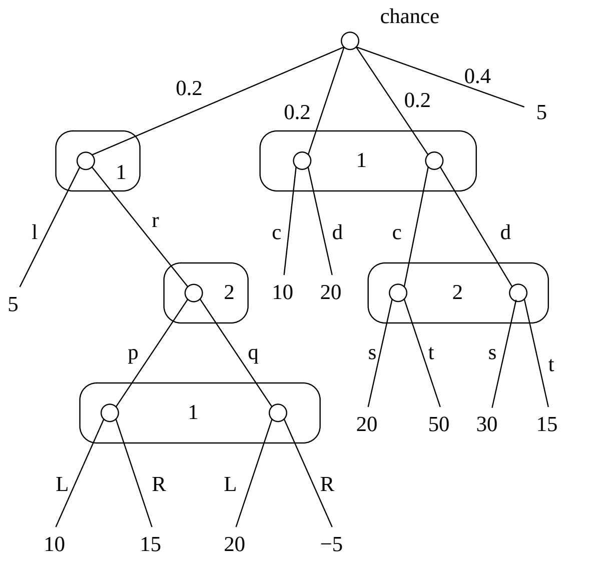


Figure 3: A zero-sum game in extensive form. (Koller, Megiddo, & von Stengel 1994)

the current state of the game. For example, in Bluff, a player always at least knows the values on her own die. We also assume *perfect recall*; that is, a player also knows the sequence of moves that have been taken so far.

The normal form equivalent does not take the structure of the extensive form's game tree. In fact, there may be many combinations of pure strategies which do not make sense to consider because they will never arise in the game. Following the example above, once player 1 chooses action r, any pure strategies including s and t are not valid because they cannot be followed anymore. Therefore, we want to use a method that captures the restrictions imposed by the structure of the game tree.

To find equilibrium strategies for an extensive form game more efficiently, the problem must be considered from a slightly different perspective. Consider the leaves of the game tree. The leaves are states which mark the end of the game and states for which payoffs are associated. Denote $\sigma^1(L)$ to be the sequence of moves taken by player 1 in a particular playout which leads to the final leaf L. Define a *realization weight* $\mu^1(\sigma^1(L))$ as the sum of probabilities over all pure strategies of the actions that can been taken in any given state which results in leaf L and would lead a sequence of $\sigma^1(L)$. Realization weights for player 2 can be defined similarly. Define $\beta(L)$ to the product of all chance nodes along the path to L. Then, if players are playing with strategies μ^1 and μ^2 then the probability that leaf L is reached is

$$\mu^1(\sigma^1(L))\mu^2(\sigma^2(L))$$

The set of linear programs can be transformed into an equivalent set simply by computing directly with the sequences rather than strategies. Define the empty sequence $\emptyset$ to be the sequence containing no moves. Each valid game sequence can then be thought of as the empty sequence or

	(p,s)	(p,t)	(q,s)	(q,t)
(l,L,c)	9	15	9	15
(l,L,d)	13	10	13	10
(l,R,c)	9	15	9	15
(l,R,d)	13	10	13	10
(r,L,c)	10	16	12	18
(r,L,d)	14	11	16	13
(r,R,c)	11	17	7	13
(r,R,d)	15	12	11	8

Table 1: The converted payoff matrix of the example extensive-form game (Koller, Megiddo, & von Stengel 1994)

a concatenation of a sequence leading to a leaf's parent and the final choice made by the player.

Constraints can be placed in the same way as before, ie. by asserting that the realization weights of all leaves sum to 1. These constraints can be defined inductively on the structure of the game tree as follows. Imagine a weight x_σ induced by an interior node whose corresponding sequence is σ. If the weight x_σ is equal to the sum of the weights of its children then the restriction that the sum at the root must equal 1 satisfies the required constraints. These constraints can be easily built by traversing the game tree. Furthermore, the number of constraints are linear in the game tree size.

The rows and columns of the payoff matrices correspond to possible sequences σ^1 and σ^2. The entries are expected payoffs: simply the product of the original payoff and chance of arriving in leaf L given the sequence: $\beta(L) \cdot a_{ij}$. Note that this matrix will be sparse because the payoff matrix entries for every pair of strategies that do not lead to a leaf will be 0. The number of leaves will be linear in the game tree size, which will reduce the effort required by the linear programming solver exponentially.

A pair of solutions to this new linear program will be the optimal weights for each node in the game tree. For each node, the distribution of the weights of valid sequences for the children describes is optimal for an equilibrium strategy. Therefore, a choice at any given node can be obtained by normalizing the distribution of these weights and choosing a child node according the this normalized distribution.

Methodology

This sections describes how we apply the general method described above to Bluff to find equilibrium strategies.

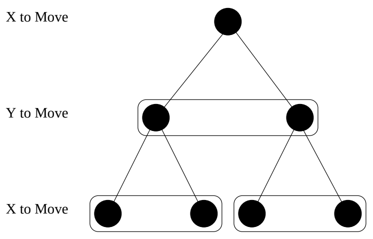
Game Tree Representation

Often, sample game trees for imperfect information games are similar to the tree depicted in Figure 4. This sort of tree is characteristic of a game where the player to move does not know the move made immediately prior by his opponent, but does know everything else about the game. The game tree for Bluff is quite different from this. In Bluff, both players have complete knowledge of all moves made during the game. The only hidden information consists of the result of the opponent's die roll at the start of the round. This re-

a	b	c	d	e	f	g	h	i	j	k	l	m
1-1	1-2	1-3	1-4	1-5	1-6	2-1s	2-2s	2-3s	2-4s	2-5s	2-6s	Bluff

Table 2: Bid-to-character encoding for all legal bids in 2-player, 1-die Bluff.

Figure 4: The standard example of the game tree of an imperfect information game. Enclosed nodes represent information sets. In this structure, the player to move does not know the immediately preceding move made by the opponent



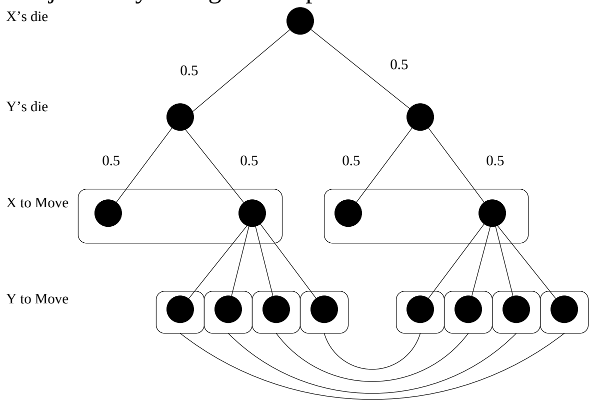
sults in a game tree that resembles Figure 5 (but which is of course much bigger and deeper).

We can therefore attach an informative label to all information sets of a player X by denoting all the information available to X at that point in the game. This information consists of X's own die roll, and the bidding sequence made thus far in the game by both X and the opponent. If we map each possible bid to single alphabetic characters in lexicographic order based on the legal bid progressions, then the label for each information set is a string consisting of the player's die roll concatenated with the bidding sequence. For example, the string X1ab is the information set for player X where X's die result is a 1, the first bid (made by X) was 1-1, and the second bid (made by Y) was 1-2. We show the encoding of all bids in Table 2, where bids are ordered according to their legal progression.

If X is the player to move first, then the length of the bidding sequence will always be odd in any information set belong to X, and odd when the information set belongs to Y. Once we have constructed information sets in this manner, we can easily denote legal moves by simply appending the character corresponding to a legal bid to the end of the string representing the information set in which that move is made.

The above construction immediately yields a way to count the exact number of leaves in the game tree, and thus also the number of non-zero entries in the payoff matrix A. Since each bid can occur at most once in a legal bidding sequence, the number of such legal sequences is clearly the number of subsets of the set $1 \ldots b-1$, where $b = n * m * f$ is simply the number of legal bids (we subtract 1 because the empty set is not a legal bid sequence). The number of subsets of an b element set is simply 2^b. Furthermore, each such bidding sequence can arise for all combinations of die rolls of both players, resulting in a final number of leaves in the game tree of $b * (2^b - 1)$. For our case, where $b = 12$, this results in 49140 game tree leaves.

Figure 5: The general structure of the game tree for bluff, in the case where each player has a 2-sided die. Enclosed nodes joined by arcing lines represent information sets.



Experimental Analysis

This section describes the empirical setup and results.

The experiments essentially traverse the game tree recursively for Bluff, filling the constraint matrices and then filling the payoff matrix once a leaf is reached. The values $\beta(L) = \frac{1}{36}$ for all leaf nodes because there is only one chance node at the beginning of the game. The tree traversal algorithm uses dynamic matrix data structures which automatically keep track of name/number associations for both rows and columns. This allows for convenient building of the matrices since the naming scheme for information sets and sequences is clear. The linear programs described in Eq. 8 and Eq. 9 from (Koller, Megiddo, & von Stengel 1994) are built from these matrices and the GNU Linear Programming Kit (GNU 2007) is used to find the solution using the simplex method. To ensure the correctness of the implementation, the algorithms were verified on normal form and sequence form versions of Rock, Paper, Scissors and the example game shown in Figure 3.

The experiments run on Bluff find equilibrium strategies for both players in sequence form. The variable `df` represents the number of die faces for both dice. The number of die faces is varied from 2 to 6. The wild value is always equal to the number of die faces, eg. with 4 die faces 4s are wild. The variables `ch` and `!ch` represent the situations where the Check move is enabled or disabled. The "value of the game" V represents the expected payoff for the first player, ie. the player that starts.

The source code and result files are available from (Lanctot & Long 2007). The source code is in the file `675proj.tar.gz`.

	V	t_{fill}	t_x	t_y	t_{tot}
`!ch, df=2`	0.5	0	0	0	$<< 1$
`!ch, df=3`	0.11111	< 1	< 1	< 1	≈ 2
`!ch, df=4`	0.0625	2	1.5	1.5	5
`!ch, df=5`	0.008	75	37	48	160
`!ch, df=6`	-0.027132	1941	977	856	3774
`ch, df=2`	0	0	0	0	$<< 1$
`ch, df=3`	0	< 1	< 1	< 1	≈ 2
`ch, df=4`	0	8	3	2	13
`ch, df=5`	0	186	57	27	270
`ch, df=6`	0	5187	1182	1033	7402

Table 3: Values and timing results for different cases of Bluff.

Results

The values of the games and timing information are listed in Table 3. t_{fill} represents the amount of time taken to fill the matrices. t_x and t_y represent the amount of time taken to find equilibrium strategies for x and y. t_{tot} represents the total time taken by the case. All times are in seconds. The experiments were run on a 64-bit dual-processor Intel Pentium 4 CPU @ 3.40 GHz.

Results are in files named like `xstrat.df4.check`; this file holds the first players' (player X's) sequence weights with the number of die faces equals to 4 and the check move enabled.

An example of the sequence-weight strategies (shown compactly, ie. only sequences of nonzero weight) computed for the case where check is disabled and the number of die faces is 2 (`!ch,df=2`) is:

```
x is                    y is
Xemptymove=1.000000     Yemptymove=1.000000
X1c=1.000000            Y1ad=1.000000
X1cde=1.000000          Y1bd=1.000000
X2d=1.000000            Y1cd=1.000000
                        Y1de=1.000000
                        Y2ad=1.000000
                        Y2bd=1.000000
                        Y2cd=1.000000
                        Y2de=1.000000
```

In this case there are only 4 possible bids, so `e` means calling Bluff. We see that for this case player 1 will always play `c` (bid 2-1s) when she has a 1 and will play `d` (bid 2-2s) whenever she has a 2. The sequence `X1cde` represents the information set where player 1 rolled a 1, bid 2-1s, player 2 bid 2-2s, and player 1 called Bluff. Likewise, the sequence `Y1ad` represents the information set where player 2 rolled a 1, player 1 bid 1-1, and player 2 bid 2-2s. The action corresponding to the Check move is always the next in sequence after Bluff. In this case, it would be `f`.

With higher die faces (d) the action encodings for Bluff and Check change. For $d = \{3, 4, 5\}$ the actions for Bluff/Check are `g`/`h`, `i`/`j`, and `k`/`l`. For the $d = 6$ case, the bid encoding is shown in Table 2.

Note that because of the constraints imposed on the weights, there are sometimes non-zero values for weights that would never arise if both players were playing equilibrium strategies. In the above example, `Y1ad` and `Y1bd` would never occur because player 1 would never play `a` or `b` first, but it is still must be part of the strategy for the constraints to hold.

The sequence-weight strategies for the `ch,df=2` case are shown below.

```
x is                    y is
Xemptymove=1.000000     Yemptymove=1.000000
X1b=1.000000            Y1ab=1.000000
X1bcd=0.500000          Y1abce=1.000000
X1bcf=0.500000          Y1abde=1.000000
X1bde=1.000000          Y1bf=1.000000
X2b=1.000000            Y1cd=0.500000
X2bcd=0.500000          Y1cf=0.500000
X2bcf=0.500000          Y1de=1.000000
X2bde=1.000000          Y2ab=1.000000
                        Y2abcf=1.000000
                        Y2abde=1.000000
                        Y2bf=1.000000
                        Y2cd=0.500000
                        Y2cf=0.500000
                        Y2de=1.000000
```

In this case, player 1 always bids 1-2 no matter what her die roll is and player 2 always calls Check in response. The rest of the values are irrelevant because they are never played.

A partial view of the sequence-weight strategies for the `!ch,df=6` case is shown below.

```
x is
X1c=0.418605  X3d=0.232558
X1d=0.372093  X3e=0.069767
X1e=0.209302  X4d=0.930233
X2c=0.418605  X4e=0.069767
X2d=0.372093  X5e=1.000000
X2e=0.209302  X6c=0.279070
X3c=0.697674  X6d=0.372093
              X6e=0.348837

y is
Y1cd=0.087209  Y3de=0.511628
Y1ce=0.387597  Y3dm=0.488372
Y1cm=0.525194  Y3ef=0.093023
Y1de=0.288760  Y3em=0.906977
Y1dm=0.711240  Y4cd=1.000000
Y1eg=0.129845  Y4de=0.162791
Y1em=0.870155  Y4dj=0.837209
Y2cd=0.174419  Y4ef=0.906977
Y2ce=0.304264  Y4em=0.093023
Y2de=0.292636  Y5ce=1.000000
Y2dm=0.707364  Y5de=1.000000
Y2eh=0.125969  Y5ek=1.000000
Y2em=0.874031  Y6cd=0.261628
Y3cd=0.087209  Y6ce=0.691860
Y3ce=0.261628  Y6ci=0.046512
Y3ci=0.651163  Y6de=1.000000
               Y6ef=1.000000
```

In the above strategies, both sequences with nonzero weight and irrelevant entries have been removed. Sequences that have more than 2 bids have been removed as well.

We see that player 1 never starts with a bid other than 1-3, 1-4, or 1-5 even when she rolled a 6. The strategies are quite mixed: player 1 only chooses deterministically when she has rolled a 5. The strategies are slightly biased towards the bid of 1-5; it seems generally that calling a bid of 1-5 is desired.

Discussion

Our results demonstrate that 1 die against 1 die Bluff, with 6-sided dice, is very slightly biased in favor of the player who plays second. However, we also see that as we play with dice with fewer faces (smaller d), the advantage shifts towards the first player to move. The reason for this is best illustrated by the case of playing with 2-sided dice ($d = 2$), where the value of the game is a whopping 0.5 for player 1. In this scenario, the first player, X, can always safely make a bid of 2-1s, because 2s are wild. Player Y cannot call Bluff, and therefore must make a bid of 2-2s. However, the chance of rolling 2-2s is only 25%, which means X is expected to win 75% of all games, resulting in the payoff listed in our results. We also note that the solution found in our results, where X always calls 2-1s when his roll is a 1 and 2-2s when his roll is a 2 is equally valid, since when X has a 2, it doesn't matter whether he calls 2-2s and lets Y call Bluff, or whether he forces Y to call 2-2s and then calls Bluff himself.

As we increase the value of d, X is faced with the fact that any bid he makes is less probable to occur, unless he simply calls the result of his own die. This can be a disastrous decision, however, if X rolls low, because Y can then simply call his own die, and X is forced to make a highly improbable bid. Since probabilistically, all bids from 1-1 to 1-$(d-1)$ are equally likely, a bid of 1-$(d-1)$ is generally a very strong bid, because it forces the other player to make a much less probable bid.

We can see this directly in our results for $d = 6$, where player X is biased towards calling 1-5, regardless of the result of his own die roll (and in fact, he always does so whenever he actually rolls a 5). In response, Y will usually call Bluff except when he has a 5 or a 6 himself, rather than be forced to make a highly improbable bid.

When Checking is added to the game, we see dramatic differences in both the value of the game and its optimal strategies. In the case where $d = 2$, Checking takes away X's ability to force Y to make the improbable bid of 2-2s, because calling 2-1s is now a losing proposition for any player that calls it. X thus loses his first player advantage and the value of the game becomes zero.

For the full value of $d = 6$, the value of the game remains zero, but the strategies are very different from the basic game. Instead of being biased towards calling 1-5 for X, so as to force Y to either call bluff or make a higher bid, Y simply never makes the improbable high bid and always calls Bluff or Check based on the roll of his die. X must then use strategies such that half the time, a call of Check will result in a win for X, and the other half, a call of Bluff will result in a win for X.

One possible reason why Checking has this effect on the game is that it ensures that the player to move always has a winning play. Without Checking, if a player X can successfully call the highest bid that is not a bluff given the dice results of both players, then X has effectively won the game, even though Y still has to make some move before the game is over. However, with Checking as an option, if Y believes that X has found the highest 'true' bid, then Y can call Check to win the game. This makes it of paramount importance to both players to not give away the result of their die rolls. Since Y cannot afford to give away information about his die, it is best to simply call bluff or check immediately. There is no need for Y to randomize over strategies, since the result of his die roll acts as a randomizer for him.

Conclusion

In this paper, we present an optimal, game-theoretic solution to the game of 2-player Bluff with 1 6-sided die per player. We found that the basic game slightly favored the second player to move, with a value of -0.027132, but that adding the Checking variant to the game resulted in a value of 0, favoring neither player.

As the size of the Bluff game tree is exponential in the number of dice, solving the game at any larger scale quickly becomes a daunting task. Furthermore, if the game is played such that a player who loses a round not only loses a die, but gives it to the other player, then solving the game seems to become a non-linear problem, since we end up with a recursive situation where the value of the 2 dice vs. 1 die game depends on the value of the 1 die vs. 2 dice game. We are aware of no known method for resolving this non-linearity, making this problem a potentially interesting direction for future research.

Acknowledgments

We thank Michael Buro for his assistance and inspiration with this project. Financial support was provided in part by NSERC and iCORE.

References

GNU. 2007. Linear programming kit. `http://www.gnu.org/software/glpk/`.

Koller, D.; Megiddo, N.; and von Stengel, B. 1994. Fast algorithms for finding randomized strategies in game trees. 750–759. `http://citeseer.ist.psu.edu/koller94fast.html`.

Lanctot, M., and Long, J. 2007. `http://www.cs.ualberta.ca/~lanctot/files/675proj/`.

Nash, J. 1950. Non-cooperative games.

von Neumann, J. 1928. Theory of parlor games.