An Investigation into the Properties of Goofspiel

David Kerr

Spring, 1992
INTRODUCTION

The two-person card game of Goofspiel, also called GOPS (game of pure strategy), proves amenable to an analysis using the theorems of modern game theory. From a standard 52-card deck, the thirteen clubs are given to player I, the thirteen diamonds to player II, the thirteen hearts are discarded, and the thirteen spades are shuffled and placed face down in the middle. From this middle pile one card is turned up. The two players then 'bid' on the upturned card, each player choosing one card and then simultaneously displaying it to the other player. The player showing the highest card (ace low, king high) wins the value of the upturned card (ace = 1, jack = 11, queen = 12, king = 13). If both players display the same card, then each player receives half the value of the card. These three cards are then discarded, a new spade from the middle pile is turned over, and the game continues. After thirteen rounds, all cards have been played. The player with the most points wins. In this case, it is the person who scores over 45.5 points (the sum of one to thirteen, divided by two).

The intuitive solution to scoring well is to play a card valued at one above what the opponent discards. However, neither player knows what the other will play. This leads to thought processes similar to the following. Say the first upturned card is a seven. Player I 'feels' as though Player II might play high for the card, and so plays a two, electing not to try to also play high. Player II 'feels' as though Player I might play a card near a seven, and so hopes to go slightly over and therefore plays a ten. Player II then wins seven points, as the ten is larger than Player I's two. However, it is not clear which player has the advantage after this round. Although Player II now has seven
points 'in the bank', Player I has higher valued cards and so is therefore likely to score more points than Player II in the following twelve rounds.

These gut feelings and second guesses about the opponent can be put into the context of game theory. Quite a few texts mention this game briefly, but only as a short example to help teach students to think in a game theoretic context. A few articles on this topic have also been published, and their results will be discussed shortly. For this analysis, let $N$ be the number of cards, and let the cards of Players I and II and the middle have values $P_1$, $P_2$, and $M_i$ respectively, where $i$ goes from 1 to $N$. In addition, $P_1 \leq P_2 \leq \ldots \leq P_N$, and $P_1 \leq P_2 \leq \ldots \leq P_N$, and $M_1 \leq M_2 \leq \ldots \leq M_N$. As stated previously, a card from the middle pile is turned up. Both players then simultaneously discard. In a slight variation on scoring, the owner of the higher card wins from the other player the value of the upturned card, or nothing in the event of a tie. These three cards are then removed, the next card from the middle is upturned, and play continues until no cards remain. The player with positive winnings at the end is declared the winner.

Let us take a quick look at a simpler problem, one where $P_1 = P_2$. This two-player game is non-cooperative, symmetric, and zero-sum. By the symmetry involved, each player's optimal strategy will be identical and result in zero profit. The strategy must be such that knowledge of it by the other player will not allow a profit to be made. Such a strategy will in all likelihood depend upon what cards have been played from both players' hands and the middle pile, and what the current upturned card is. It might be pure, meaning that a particular card should always be played, or it might be mixed, meaning that two or more cards each have a positive probability of being played. This optimal strategy must maximize the expected profit to both players; by symmetry, it gives each player an expected profit of zero.
OPTIMAL STRATEGY AGAINST A RANDOMIZING OPPONENT - ROSS

Ross [7] is one of the few to publish an in-depth look at Goofspiel. In his paper, he demonstrates three points: if Player II is playing cards randomly, Player I's optimal strategy is to match the upturned card; if both players must discard before the middle card is overturned, then the randomizing strategy is optimal for both players; and Goofspiel is a stochastic game, with interesting recursive properties for solving.

Theorem. If Player II discards randomly (with uniform probability over all remaining cards), then Player I's optimal strategy is to discard the card having value $P_1i$ when the value of the upturned middle card is $M_i$.

Proof. An inductive proof on $N$ is used. The theorem is trivial for $N = 1$, so assume it true for $N - 1$. Suppose that for the $N$-card problem the first upturned card from the middle has value $M_j$, and investigate any strategy which calls for Player I to play $P_1i$ where $i < j$. After the first round, Player I has cards $1, \ldots, i-1, \ldots, j, \ldots, N$, whereas the middle pile has cards $1, \ldots, i, \ldots, j-1, \ldots, N$. But now, from the induction hypothesis, it follows that of the strategies that have Player I play $P_1i$, the best (call it Strategy 1) is the one which on the following plays discards

$$
\begin{align*}
P_1k & \quad \text{on } M_k, \quad k = 1, \ldots, i-1 \\
\ 
\end{align*}
$$

$$
\begin{align*}
P_{1k+1} & \quad \text{on } M_k, \quad k = i, \ldots, j-1 \\
\ 
\end{align*}
$$

$$
\begin{align*}
P_1k & \quad \text{on } M_k, \quad k = j+1, \ldots, N. \\
\ 
\end{align*}
$$

However, compare this to a new strategy (call it Strategy 2) which is the same as Strategy 1 above with the exception that it uses:
$P_{1_{i+1}}$ on $M_j$
$P_{1_i}$ on $M_i$

whereas Strategy 1 uses:

$P_{1_i}$ on $M_j$
$P_{1_{i+1}}$ on $M_i$.

The expected winnings to Player I looking only at these two plays of Strategy 1 is, by checking the probability of Player I winning the upturned card:

$$\frac{1}{N} M_j \left( \left( \text{Number } k : P_{2_k} < P_{1_i} \right) - \left( \text{Number } k : P_{2_k} > P_{1_i} \right) \right)$$
$$+ \frac{1}{N} M_i \left( \left( \text{Number } k : P_{2_k} < P_{1_{i+1}} \right) - \left( \text{Number } k : P_{2_k} > P_{1_{i+1}} \right) \right).$$

The expected winnings to Player I for these two plays using Strategy 2 is:

$$\frac{1}{N} M_j \left( \left( \text{Number } k : P_{2_k} < P_{1_i} \right) - \left( \text{Number } k : P_{2_k} > P_{1_i} \right) \right)$$
$$+ \frac{1}{N} M_i \left( \left( \text{Number } k : P_{2_k} < P_{1_{i+1}} \right) - \left( \text{Number } k : P_{2_k} > P_{1_{i+1}} \right) \right).$$

Because $M_j \geq M_i$ and the second bracketed term in each expression is at least as large as the first bracketed term, Strategy 2 yields at least as large a payoff as Strategy 1. Therefore, for any $i < j$, when the initial upturned card is $M_j$, there exists a strategy which plays $P_{1_{i+1}}$ that is as good as any that plays $P_{1_i}$ initially. By repeating this argument, it follows that there is a strategy that plays $P_{1_j}$ that is at least as good as any playing $P_{1_i}$. Similar results are obtained in the case $i > j$, showing that there is a strategy that plays $P_{1_j}$ that is again at least as good as any playing $P_{1_i}$. Therefore, by the induction hypothesis, the optimal strategy against a randomizing opponent is to always match the index of the upturned card.

The expected payoff in such a case is
\[ \frac{1}{N} \sum_{j} M_j \left[ (\text{Number } j : P_{2j} < P_{1i}) - (\text{Number } j : P_{2j} > P_{1i}) \right]. \]

(Note: all summations go from 1 to N unless otherwise specified.) In the case where \( P_{1i} = P_{2i} \), the expected payoff is

\[ \frac{1}{N} \sum_{i} M_i [2i - N - 1] \]

In the case, \( P_{1i} = P_{2i} = M_i = i \), the payoff is

\[ \frac{1}{N} \sum_{i} [(2i^2 - Ni - i) = \frac{(2N + 1)(N + 1)}{3} - \frac{(N)(N + 1)}{2} - \frac{(N + 1)}{2} \]

\[ = \frac{(N - 1)(N + 1)}{6}. \]

In the standard 13-card case, winnings of 28 are expected.

MATCH AGAINST RANDOMIZING OPPONENT - DROR

Dror [1] has published a much more concise proof of the above theorem. As before, assume Player II discards completely randomly. Once again, prove that the strategy maximizing Player I's expected winnings is the one that has Player I discard \( P_{1i} \) on middle card \( M_i \).

Proof. Let \( a_i \) and \( b_i \) be indexed from 1 to N. Let vectors \( \alpha \) and \( \beta \) be any two permutations of \( (1, 2, ..., N) \). Define vectors \( a_\alpha = (a_{\alpha(1)}, a_{\alpha(2)}, ..., a_{\alpha(N)}) \) and \( b_\beta = (b_{\beta(1)}, b_{\beta(2)}, ..., b_{\beta(N)}) \). Denote the scalar product

\[ a_\alpha \cdot b_\beta = \sum_i a_{\alpha(i)} * b_{\beta(i)}. \]

Then
\[
\max_{\alpha, \beta} \cdot b_\beta = \sum_i a_i \cdot b_i, \quad \text{where } a_1 \leq a_2 \leq \ldots \leq a_N; b_1 \leq b_2 \leq \ldots \leq b_N.
\]

Denote the probability that Player I wins the trick given a discard of \(P_1i\) to be \(\pi_i\). Clearly, \(\pi_1 \leq \pi_2 \leq \ldots \leq \pi_N\). These probabilities are independent of the upturned card \(M_i\). Thus the scalar product of the ordered sequence of the vector of middle cards \((a_\alpha = M_i)\) with the vector of win probabilities \((b_\beta = \pi_i)\) takes the maximal value when matched in order.

**HIDDEN CARD GOOFSPIEL - ROSS**

Ross examines another variation on the basic game, Hidden Card Goofspiel, in which both players discard before the card from the middle is overturned and its point value revealed. In this deviation, where both players are bidding on a card whose value they don’t know, the optimal strategy for both is to randomize.

**Theorem.** For Hidden Card Goofspiel, randomizing is optimal for each player and the expected profit of the game to Player I is

\[
\frac{1}{N^2} \sum_k M_k \sum_{i,j} \varphi(P_{1i}, P_{2j}), \quad \text{where}
\]

\[
\varphi(i,j) = \begin{cases} 
1, & \text{if } i > j \\
0, & \text{if } i = j \\
-1, & \text{if } i < j.
\end{cases}
\]

**Proof.** The proof is obtained by demonstrating that Player I’s expected profit using randomization is as above, regardless of Player II’s strategy. An induction argument on \(N\) is used. The theorem is trivially true for \(N = 1\), so
assume it true for \( N - 1 \). Suppose for the \( N \)-card case, Player II initially plays \( P_{2j} \). Then it follows, by the induction hypothesis, that Player I's expected payoff using randomization is

\[
1/N^2 \sum_{i,k} [M_k \partial(P_{1i},P_{2j}) + 1/(N - 1)^2 \sum M_l \sum \partial(P_{1l},P_{1m})].
\]

So, it must be shown that the above two formulae are equal. The first is the expected payoff to Player I given that both players randomize. (In the case where the two players have the same cards, this is equal to zero.) However, by writing the first equation as the winnings to Player I on the play in which Player II discards \( P_{2j} \) plus the winnings to Player I on the remaining plays of the game, as well as by conditioning on the point value of the upturned card and Player I's discard, it is seen that the second equation also represents the expected payoff to Player I given that both players randomize. So the two formulae equate, and the induction is complete. So both players can expected zero profit by randomizing.

GOOFSPIEL AS SUPER-GAME - ROSS

Ross concludes with an analysis of Goofspiel as a super-game. He claims that the number of pure strategies for each player is

\[
N-1
\]

\[
(\text{NN}) \prod_{k} k^{(k)(k+1)}.
\]

Ross says this is true because, "for each initial upturned middle card, Player I has a choice of \( N \) cards; hence, the first term \( \text{NN} \)." Conditional on the initial upturned card and the first card played by Player I, the choice for the second
card played is determined by the second upturned card and the first card played by Player I. This reasoning progresses until with one card remaining Player I has no choice of what to play.

Ross' argument seems flawed. If Player I has a choice of \( N \) cards for each initial upturned middle card, the first term should be \( N^2 \), not \( N^N \). It seems more logical that the number of pure strategies would equal

\[
\frac{(N-1)}{(N \cdot N) \prod [k \cdot k \cdot (k+1)] = (N!)^3}
\]

A partial diagram of the strategies for the case \( N = 3 \) is reprinted here from Owen [6]. It would seem to also disagree with Ross' formula.
This function gets large extremely fast: for \( f(2) = 8, f(3) = 216, \)
\( f(4) = 13,824 \). It is therefore not feasible to write down all possible pure
strategies and calculate the payoffs. Instead, the problem should be examined
as a super-game consisting of \( N \) subgames. Let

\[
f(P_{11}, ..., P_{1N}; P_{21}, ..., P_{2N}; M_1, ..., M_N; M_k)
\]

be the expected payoff to Player I, who initially has cards \( P_{1i} \), playing against
Player II who has cards \( P_{2i} \), with the pile in the middle containing cards \( M_i \),
with \( M_k \) initially upturned. In this case then, set the above function equal to
the value of the \( N \times N \) game with payoff matrix \([X(i,j)]\), where

\[
X(i,j) = M_k \times f(P_{1i}, P_{2j}) + \frac{1}{(N - 1)} \sum_{l \neq k} f(P_{11}, ..., P_{1i-1}, P_{1i+1}, ..., P_{1N};
                              P_{21}, ..., P_{2j-1}, P_{2j+1}, ..., P_{2N};
                              M_1, ..., M_{k-1}, M_{k+1}, ..., M_N; M_l)
\]

This is merely writing the payoff to Player I as the outcome of the initial play of
the \( N \)-card problem added to the average value of the payoff for all subsequent
\((N-1)\)-card problems. This recursive algorithm results in the need to now only
solve

\[
j \times (N \text{ choose } j)^3 \quad j \times j \text{ game matrices for } j = 2, ..., N.
\]

For example, the three-card case would require solving 3 three-by-three and 54
two-by-two games. The four-card problem requires 4 four-by-four, 192 three-by-
three, and 432 two-by-two games. Later, it will be shown that the number of
two-by-two games required to actually be calculated can be reduced in half.
If the order of the cards is pre-set and known to both players, the number of matrices to be solved drops, and now one must only recursively solve

\[(N \text{ choose } j)^2 \quad j \times j \text{ game matrices for } j = 2, \ldots, N.\]

The three-card problem now only needs 1 three-by-three and 9 two-by-two games solved. The four-card problem requires 1 four-by-four, 16 three-by-three, and 36 two-by-two games computed.

SOLVING THE TWO- AND THREE-CARD PROBLEMS

Using Ross' approach, what then would the optimal strategy be for the standard 13-card problem? (For the remainder of this discussion, assume \(P_{1i} = P_{2i} = M_i\). Also assume that all cards are non-negative and no two cards in a hand are identical. A 'standard' case is defined as one having \(P_{1i} = P_{2i} = M_i = i\).) A look at smaller cases proves instructive. The one-card problem is trivial, neither player has any choice and both are guaranteed a payoff of zero. The standard two-card problem is also straightforward, but now a choice must be made. Different decisions will be made, conditional upon the initial card upturned. The two relevant matrices, in terms of payoff to Player I and with Player I's first card played indexed on the left and Player II's first card played indexed on the top, are

\[
\begin{array}{cc}
1 & 2 \\
1 & 0 & 1 \\
2 & -1 & 0
\end{array}
\quad \begin{array}{cc}
1 & 2 \\
1 & 0 & -1 \\
2 & 1 & 0
\end{array}
\]

'1' is 1st card up '2' is 1st card up
So a pure strategy for Player I exists, which will guarantee at least a payoff of zero. This optimal strategy is to match the upturned card; to play a ‘1’ when the initial card is a ‘1’, and likewise play a ‘2’ if the initial card is a ‘2’. The symmetry of the game applies to Player II, who will follow suit, resulting in a payoff of zero to both. Deviating from the strategy can only be detrimental. This result holds true for non-standard two-card cases as well; both players will match the upturned cards.

A similar situation occurs in the standard three-card problem. The computations are not as obvious, however. Each of the nine elements in the three matrices (corresponding to the three possible initial upturned cards) must be calculated by adding the payoff of the first play to the expected payoff of the subsequent two plays. The value of the subsequent plays is the average value of the payoffs of the subgames determined by each of the possible next two upturned cards. If on the first play, Player I discards P1_i and Player II discards P2_j, and the initial middle card is M_k with M_l and M_m still face down in random order, the expected payoff of the game is

\[ M_k \times \delta(P1_i, P2_j) + \frac{1}{2} \text{ [ Payoff using remaining cards if } M_l \text{ is next card upturned ]} + \frac{1}{2} \text{ [ Payoff using remaining cards if } M_m \text{ is next card upturned ]}. \]

As a computational aid, it can be shown that, in general, the last two terms of this formula are equal; that is, the order of the last two cards in the middle does not affect the expected payoff to Player I. For succinctness, let Player I’s remaining two cards have values ‘a’ and ‘b’, while Player II still retains cards of value ‘c’ and ‘d’, and the middle contains cards valued at ‘l’ and ‘m’. The two following matrices correspond to the value of the next
upturned card, and are indexed by the two possible plays of each player on this next-to-last round.

\[
\begin{align*}
& \text{c} & \text{d} \\
\text{a} & 1 \cdot \partial(a, c) + m \cdot \partial(b, d) & 1 \cdot \partial(a, d) + m \cdot \partial(b, c) \\
\text{b} & 1 \cdot \partial(b, c) + m \cdot \partial(a, d) & 1 \cdot \partial(b, d) + m \cdot \partial(a, c)
\end{align*}
\]

'\text{c}' is 2nd to last card up

\[
\begin{align*}
& \text{c} & \text{d} \\
\text{a} & m \cdot \partial(a, c) + 1 \cdot \partial(b, d) & m \cdot \partial(a, d) + 1 \cdot \partial(b, c) \\
\text{b} & m \cdot \partial(b, c) + 1 \cdot \partial(a, d) & m \cdot \partial(b, d) + 1 \cdot \partial(a, c)
\end{align*}
\]

'm' is 2nd to last card up

These two matrices have the form \( \begin{pmatrix} A & B \\ C & D \end{pmatrix} \) and \( \begin{pmatrix} D & C \\ B & A \end{pmatrix} \) so the payoffs generated by these two matrices are the same, because the two are identical through transposing the rows and columns, which does not affect the payoff. Therefore, with two cards remaining in each player's hand, only one of the two possible remaining middle cards need be examined to determine the resulting values. The two by two matrix might be solved by pure or mixed strategies and its payoff calculated accordingly.

In the standard three-card problem, the matrices look as follows
Some interesting properties emerge. As expected, the matrices are skew-symmetric, with zeroes along the diagonals. If the '1' or the '3' is the initial card upturned, the optimal strategy is a pure one; both players will match the upturned card. As the final case, if the '2' is the first card upturned, then a type of equilibrium is seen. Both players are indifferent between playing their '2' and '3' cards. So even before the first middle card is flipped over, both players know that they will match it with their respective cards. In the second round, both players will also match the next upturned card. By default, the final round will also see both players match the last middle card. Any deviations from this strategy will lead to negative expected payoffs. This same strategy of both players matching the upturned card is trivially true in the one- and two-card problem. But is the strategy of matching the initial upturned card optimal for the N-card problem?

SOLVING THE FOUR-CARD PROBLEM

Using the same approach as above, the solution to the four-card problem can be found. If Player I initially discards P1i and Player II plays P2j on the first middle card Mk with Ml, Mm, and Mn still turned down in the pile, the payoff to Player I is

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.00</td>
<td>0.40</td>
<td>2.80</td>
</tr>
<tr>
<td>2</td>
<td>-0.40</td>
<td>0.00</td>
<td>0.40</td>
</tr>
<tr>
<td>3</td>
<td>-2.80</td>
<td>-0.40</td>
<td>0.00</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.00</td>
<td>-1.00</td>
<td>1.25</td>
</tr>
<tr>
<td>2</td>
<td>1.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>3</td>
<td>-1.25</td>
<td>0.00</td>
<td>0.00</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.00</td>
<td>-2.00</td>
<td>-0.67</td>
</tr>
<tr>
<td>2</td>
<td>2.00</td>
<td>0.00</td>
<td>-2.00</td>
</tr>
<tr>
<td>3</td>
<td>0.67</td>
<td>2.00</td>
<td>0.00</td>
</tr>
</tbody>
</table>
\[ M_k \ast \partial(P_1, P_2) + \\
1/3 \left( \text{Payoff using remaining cards if } M_1 \text{ is next card upturned} \right) + \\
1/3 \left( \text{Payoff using remaining cards if } M_m \text{ is next card upturned} \right) + \\
1/3 \left( \text{Payoff using remaining cards if } M_n \text{ is next card upturned} \right). \]

Unfortunately, unlike the two-card case, these three final terms are not necessarily equal (although they are typically quite close). Therefore each of these last terms must be calculated out. Each term is the solution of a three by three payoff matrix, each element of which is the sum of two parts; the result of the second round of cards and the solution of the two by two matrix for the final two rounds using the remaining cards. A Pascal program was used to numerically approximate the payoffs of the three by three matrices of the subgames. Roundoff error makes the entries in the four matrices accurate to approximately 0.01.

\[
\begin{array}{cccc}
1 & 2 & 3 & 4 \\
1 & 0.00 & 0.02 & 1.97 & 4.25 \\
2 & -0.02 & 0.00 & 0.16 & 2.54 \\
3 & -1.97 & -0.16 & 0.00 & 0.62 \\
4 & -4.25 & -2.54 & -0.62 & 0.00 \\
\end{array} \quad \begin{array}{cccc}
1 & 2 & 3 & 4 \\
1 & 0.00 & -1.19 & 0.31 & 3.15 \\
2 & 1.19 & 0.00 & -0.76 & 2.28 \\
3 & -0.31 & 0.76 & 0.00 & -0.18 \\
4 & -3.15 & -2.28 & 0.18 & 0.00 \\
\end{array}
\]

'1' is 1st card \quad '2' is 1st card

\[
\begin{array}{cccc}
1 & 2 & 3 & 4 \\
1 & 0.00 & -2.00 & -0.67 & 1.58 \\
2 & 2.00 & 0.00 & -2.00 & 0.67 \\
3 & 0.67 & 2.00 & 0.00 & -0.82 \\
4 & -1.58 & -0.67 & 0.82 & 0.00 \\
\end{array} \quad \begin{array}{cccc}
1 & 2 & 3 & 4 \\
1 & 0.00 & -3.29 & -2.08 & -0.30 \\
2 & 3.29 & 0.00 & -3.17 & -1.20 \\
3 & 2.08 & 3.17 & 0.00 & -2.61 \\
4 & 0.30 & 1.20 & 2.61 & 0.00 \\
\end{array}
\]

'3' is 1st card \quad '4' is 1st card
Unlike the earlier cases, a strategy of always matching the initial upturned card is not optimal. In the cases of a '2' or '3' initially up, Player II could respond by playing the card valued at one higher than that of the upturned card, resulting in Player I making negative expected winnings. It is still true that an initial '1' or '4' results in a matching strategy being optimal the first round, but the strategies used on an initial '2' or '3' are mixed. Using Mathematica, the exact values of this strategy can be found.

<table>
<thead>
<tr>
<th>if '2' is 1st card</th>
<th>if '3' is 1st card</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prob (play '1')</td>
<td>0.336</td>
</tr>
<tr>
<td>Prob (play '2')</td>
<td>0.137</td>
</tr>
<tr>
<td>Prob (play '3')</td>
<td>0.527</td>
</tr>
<tr>
<td>Prob (play '4')</td>
<td>0.000</td>
</tr>
<tr>
<td></td>
<td>0.269</td>
</tr>
<tr>
<td></td>
<td>0.000</td>
</tr>
<tr>
<td></td>
<td>0.513</td>
</tr>
<tr>
<td></td>
<td>0.218</td>
</tr>
</tbody>
</table>

SOLVING THE FIVE-CARD PROBLEM

As a slightly more complex problem, the standard five-card case can be investigated. However, the values of numerous four by four payoff matrices must now be approximated by computer, an arduous task. Theoretically, 500 four by four matrices must be evaluated, but if the knowledge that the five by five matrix is skew-symmetric is used, this number is reduced to 200. In addition, 3000 three by three matrices and over 200,000 two by two matrices are evaluated. The general procedure is as done earlier; however, precision was sacrificed in order to make the program feasible to run. For this reason, the entries have an error of 0.2.
<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.0</td>
<td>-0.2</td>
<td>1.4</td>
<td>3.2</td>
<td>5.7</td>
</tr>
<tr>
<td>2</td>
<td>0.2</td>
<td>0.0</td>
<td>-0.1</td>
<td>1.8</td>
<td>4.4</td>
</tr>
<tr>
<td>3</td>
<td>-1.4</td>
<td>0.1</td>
<td>0.0</td>
<td>0.2</td>
<td>2.9</td>
</tr>
<tr>
<td>4</td>
<td>-3.2</td>
<td>-1.8</td>
<td>-0.2</td>
<td>0.0</td>
<td>0.8</td>
</tr>
<tr>
<td>5</td>
<td>-5.7</td>
<td>-4.4</td>
<td>-2.9</td>
<td>-0.8</td>
<td>0.0</td>
</tr>
</tbody>
</table>

'1' is 1st card

1 2 3 4 5

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>-1.5</th>
<th>-0.2</th>
<th>1.9</th>
<th>4.7</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1.5</td>
<td>0.0</td>
<td>-1.1</td>
<td>1.2</td>
<td>4.0</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0.2</td>
<td>1.1</td>
<td>0.0</td>
<td>-0.5</td>
<td>2.3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>-1.9</td>
<td>-1.2</td>
<td>0.5</td>
<td>0.0</td>
<td>-0.2</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>-4.7</td>
<td>-4.0</td>
<td>-2.3</td>
<td>0.2</td>
<td>0.0</td>
<td></td>
</tr>
</tbody>
</table>

'2' is 1st card

1 2 3 4 5

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>-3.4</th>
<th>-2.2</th>
<th>-0.6</th>
<th>1.9</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>3.4</td>
<td>0.0</td>
<td>-3.2</td>
<td>-1.4</td>
<td>1.3</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2.2</td>
<td>3.2</td>
<td>0.0</td>
<td>-2.8</td>
<td>0.1</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0.6</td>
<td>1.4</td>
<td>2.8</td>
<td>0.0</td>
<td>-1.8</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>-1.9</td>
<td>-1.3</td>
<td>-0.1</td>
<td>1.8</td>
<td>0.0</td>
<td></td>
</tr>
</tbody>
</table>

'3' is 1st card

1 2 3 4 5

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>-4.5</th>
<th>-3.5</th>
<th>-2.0</th>
<th>0.2</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>4.5</td>
<td>0.0</td>
<td>-4.4</td>
<td>-2.8</td>
<td>-0.6</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3.5</td>
<td>4.4</td>
<td>0.0</td>
<td>-4.0</td>
<td>-1.7</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>2.0</td>
<td>2.8</td>
<td>4.0</td>
<td>0.0</td>
<td>-3.4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>-0.2</td>
<td>0.6</td>
<td>1.7</td>
<td>3.4</td>
<td>0.0</td>
<td></td>
</tr>
</tbody>
</table>

'4' is 1st card

1 2 3 4 5

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>-4.5</th>
<th>-3.5</th>
<th>-2.0</th>
<th>0.2</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>4.5</td>
<td>0.0</td>
<td>-4.4</td>
<td>-2.8</td>
<td>-0.6</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3.5</td>
<td>4.4</td>
<td>0.0</td>
<td>-4.0</td>
<td>-1.7</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>2.0</td>
<td>2.8</td>
<td>4.0</td>
<td>0.0</td>
<td>-3.4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>-0.2</td>
<td>0.6</td>
<td>1.7</td>
<td>3.4</td>
<td>0.0</td>
<td></td>
</tr>
</tbody>
</table>

'5' is 1st card

As before, Mathematica can give an exact listing of the optimal strategies,
<table>
<thead>
<tr>
<th></th>
<th>'1'</th>
<th>'2'</th>
<th>'3'</th>
<th>'4'</th>
<th>'5'</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prob (play '1')</td>
<td>0.06</td>
<td>0.19</td>
<td>0.12</td>
<td>0.11</td>
<td>0.11</td>
</tr>
<tr>
<td>Prob (play '2')</td>
<td>0.82</td>
<td>0.00</td>
<td>0.12</td>
<td>0.07</td>
<td>0.04</td>
</tr>
<tr>
<td>Prob (play '3')</td>
<td>0.12</td>
<td>0.73</td>
<td>0.00</td>
<td>0.22</td>
<td>0.00</td>
</tr>
<tr>
<td>Prob (play '4')</td>
<td>0.00</td>
<td>0.08</td>
<td>0.75</td>
<td>0.18</td>
<td>0.00</td>
</tr>
<tr>
<td>Prob (play '5')</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.42</td>
<td>0.85</td>
</tr>
</tbody>
</table>

Some interesting patterns emerge from studying the above solution. In general, on the first play of the game the players discard a card valued at one higher than the upturned card. Very rarely will they match the upturned card (except in the case that the highest valued card is upturned). This is quite different than the three-card problem, where both players match every upturned card. The solution to the four-card problem is a mixture of these two. As an extrapolation to larger N-card problems, it seems likely that an optimal strategy on the play of an intermediated valued initial card would be approximately 70% to play one value higher, 10% to play two values higher, 10% to play the '1', and 10% to play the '2'. As a subject for further study, the analysis of the standard card problem for these larger cases of $N$ could be done. A possible solution to such a problem would be to link a recursive program in a high-level language with the mathematical power of a linear programming application, such as Mathematica or Lindo, among others.

**CARD CONSTRAINTS WHICH ALLOW PERMANENT MATCHING TO BE OPTIMAL**

Under what conditions can Player I always match the middle cards, without using any knowledge about the actions of Player II? This is true in any two-card problem, as well as the standard three-card problem, but neither the standard four- nor five-card problems. As a first step, it can be checked
whether the matching strategy is optimal for all three-card games, where the values of the three cards are arbitrary (but the same for both players and the middle).

Theorem. Take the three-card problem with each player and the middle containing cards valued at c1, c2, and c3. Without loss of generality, say 0 < c1 < c2 < c3. If c3 ≥ c2 + c1, the strategy of always playing a card equal in value to the upturned card is optimal.

Proof. This is a symmetric game, so the payoff for the optimal strategy is zero. If Player I announces "I will always match the upturned card," is there any strategy for Player II that will give Player I negative expected payoffs? The answer is no. If Player II fails to match during the round that c3 is upturned, then Player I is guaranteed of at least zero (c3 - c2 - c1) winnings. So Player II must match c3 along with Player I. So this simplifies to the two-card case, where it has been seen that Player II will again match the upturned cards, deviation resulting in a loss valued at c2 - c1. Therefore, both players will match all three upturned middle cards.

Theorem. Take the labeling and ordering of the cards as above. If the strategy of always matching the upturned card is optimal, then c3 ≥ c1 + c2. (This is the converse of the above theorem, making the two conditions an 'if and only if' relationship.)

Proof. If c(i) is the initial upturned card, it must be seen under what conditions Player I makes non-negative payoffs by matching c(i), regardless of Player II's first play. Let f(P1i, P2j, Mk) represent the expected payoff to Player
I if Player I and Player II discard 'i' and 'j' on the initial middle card 'k'.

Assume Player I always matches the middle card, and Player II knows this and plays optimally using this knowledge. We know \( f(c_1, c_1, c_1) = f(c_2, c_2, c_2) = f(c_3, c_3, c_3) = 0 \). The payoffs for the remaining cases are

\[
\begin{align*}
    f(c_1, c_2, c_1) &= -c_1 + \min(c_2, c_3 - c_2) = \min(c_2, c_3-c_2) \\
    f(c_1, c_3, c_1) &= -c_1 + \min(c_3, c_3 + c_2) = c_3 - c_1 > 0 \\
    f(c_2, c_1, c_2) &= c_2 + \min(-c_1, c_3 - c_1) = c_2 - c_1 > 0 \\
    f(c_2, c_3, c_2) &= -c_2 + \min(c_3, c_3 - c_1) = c_3 - c_1 - c_2 \\
    f(c_3, c_1, c_3) &= c_3 + \min(-c_1 - c_2, -c_1) = c_3 - c_1 - c_2 \\
    f(c_3, c_2, c_3) &= c_3 + \min(-c_2, c_3 - c_2) = c_3 - c_2 > 0
\end{align*}
\]

For those payoff functions which are not already constrained to be greater than or equal to zero to be so, the condition must hold that \( c_3 \geq c_1 + c_2 \). For the matching card strategy to be optimal, this condition must be met (as it is for the standard three-card problem).

The conditions that must exist such that Player I can match cards throughout the game without thinking and make non-negative winnings can be expanded to an N-card situation. The cards, \( c_1, c_2, ..., c_N \), (with \( c_1 < c_2 < ... < c_N \)) must be such that

\[
c(i) \geq \sum_{k} c(k) \text{ for } i = 2, 3, ..., N.
\]

As illustrated above in the three-card case, if this condition does not hold true for card \( c(i) \), then Player II could use the strategy:

\[
\begin{align*}
    c(k+1) &\text{ on } c(k) \text{ for } k = 1, 2, ..., i-1 \\
    c(1) &\text{ on } c(i) \\
    c(k) &\text{ on } c(k) \text{ for } k = i+1, ..., N
\end{align*}
\]
and get winnings equal to

\[\sum_{k} c(k) - c(i) > 0.\]

CARD CONSTRAINTS WHICH ALLOW PERMANENT MATCHING UNDER THREAT OF PLAYING OPTIMALLY IF DEVIATION OCCURS.

Let the situation be changed so that Player I now plays with slightly more intelligence. Instead of always matching the upturned card regardless of the activities of Player II, let Player I inform Player II, "I will continue to match the upturned cards, as long as you do likewise. However, if you deviate from matching the middle cards, I will play optimally from that point onward." A new investigation into what combination of cards would make this 'matching until deviation' strategy optimal is warranted.

Once again, the one- and two-card cases are trivial. Any possible ranking of the cards will give the desired results. The three-card case is more problematic, however. It must be true that Player II will also agree to match the initial upturned card. If Player II deviates the first round, it must be that the expected retaliation by Player I will more than make up for any advantage that Player II received. If both players match the first round, then it has already been shown that both will match the final two cards as well. Once again, the three cards that each player and the middle pile contain are \(c_1, c_2, \text{ and } c_3\), such that \(0 < c_1 < c_2 < c_3\).

Theorem. Take the labeling and ordering of the cards as above. If the 'matching until deviation' strategy is optimal, then \(c_3 \geq c_1 + c_2\).
Proof. This is the same result as the previous section, but the proof is slightly more difficult. We know \( f(c1, c1, c1), f(c2, c2, c2), \) and \( f(c3, c3, c3) \) are all equal to zero, regardless of the card values. So it must be checked that the other six possibilities also give positive or zero winnings. This will just be explicitly shown for \( f(c2, c3, c2) \), the others follow in similar vein. This particular case is the situation wherein Player II plays over Player I when the initial card is \( c2 \). Under what conditions of \( c(i) \) will the expected winnings to Player I in this situation be non-negative? Depending upon the relative magnitude of \( c3 \), the final two rounds might have a pure strategy or mixed strategy, leading to the following formula for \( f(c2, c3, c2) \):

\[
\begin{align*}
c3 - c2 - c1 & \quad , \text{ if } 2 * c1 \leq c3 \\
\left[ \left( (c3)^2 + (c1)^2 - (c3 * c1) \right) / (c3 + c1) \right] - c2 & \quad , \text{ if } 2 * c1 > c3.
\end{align*}
\]

Looking at the first case, we see that if \( c3 \geq c1 + c2 \), then the function value is non-negative. Is it possible for the result in the second case to ever be non-negative? No, because this would mean

\[
\left[ \left( (c3)^2 + (c1)^2 - (c3 * c1) \right) / (c3 + c1) \right] \geq c2,
\]

\[
(c3 - c1)^2 / (c3 + c1) \geq c2 - [(c3 * c1) / (c3 + c1)],
\]

\[
(c3 - c1)^2 \geq (c3 * c2) + (c2 * c1) - (c3 * c1).
\]

But the conditions of the problem tell that \( 2 * c1 > c3 \) and \( c2 > c1 \), which implies

\[
(c1)^2 > (c3 - c1)^2 \text{ and }
\]

\[
(c3 * c2) + (c2 * c1) - (c3 * c1) > (c2 * c1) > (c1)^2,
\]
therefore \((c1)^2 > (c1)^2\). So there are no solutions to \(c3, c2,\) and \(c1\) which give non-negative winnings when \(c3 < 2 \times c1\). So for \(f(c2, c3, c2)\) to not be negative, \(c3 \geq c1 + c2\). This same general procedure can be applied to \(f(c1, c2, c1)\) which again gives the result that \(c3 \geq c1 + c2\) (here the split in the function occurs at \(c3 = 2 \times c2\)). If we assume that \(c3 \geq c1 + c2\), then it can be easily shown that \(f(c2, c1, c2), f(c3, c1, c3), f(c1, c3, c1),\) and \(f(c3, c2, c3)\) are always strictly positive. So only given the combination of cards, \(c3 \geq c1 + c2\), will Player II also match the upturned cards against a Player I who threatens optimal play should Player II deviate from matching.

The same card constraints were obtained for the three-card problem using both the ‘permanent matching’ and ‘matching until deviation’ strategies. Is this true for the larger \(N\)-card cases as well? No proof is included here, but various computational tests make it appear so. So Player I’s seemingly better strategy works only for the same set of cards which allow the more simple-minded strategy to succeed. This seems somewhat strange; it might have been expected that ‘matching until deviation’ would work with a more lenient set of cards, but this appears not to be the case.

**CONCLUSION**

At first glance, Goofspiel seems a relatively simple children’s card game. As has been seen, there is much more to this game. One facet is to test simple strategies and see under what conditions they prove optimal, or find the strategies that would beat them. A computational project is to expand the game to its proper 13-card case, and look for any patterns that develop.

As mentioned earlier, the specific order of the cards seems to only slightly affect the resulting payoffs of the subgames, so perhaps a given, fixed
order of cards can be presumed without hurting the accuracy of the computations terribly much. Ross' formula describing the number of calculations in such a situation shows a very helpful improvement in computability would be the result.

It would be interesting to see if any economic or industrial applications can be represented by Goofspiel. For example, it might be possible to model the activities of a duopsony (two firms competing to buy a particular good) by this method. Another example might be two architectural firms with finite resources competing for a series of projects, where each must decide the amount of time and money it should spend preparing plans for each job, and where the company with the best design gets that particular job. A similar situation could exist with two generals deciding on the number of troops to send to each in a series of battles.

The 'surface' of the payoff matrix is interesting to look at. It makes one wonder what it would look like if the players' choices could take on a continuum of values; in other words, make the game continuous, as opposed to discrete. Such a game might have each player given an initial allotment to use, but the division of this into N amounts is left to the discretion of each player.

There are several variations to the base game presented here that would be interesting to explore. Three person Goofspiel can also be played, and would be quite a bit more challenging to investigate. Such a game would have the added dimensions of conspiracies and contracts between the players. Another possible version would have both players able to see the upturned card, but not what the opponent plays. A judge would simply tell each player who won each particular round. In this situation, players would not know exactly which cards were contained in the opponent's hand after the first round, making
optimal strategy more difficult to analyze. An interesting asymmetric problem would involve the case where Player I knows the exact ordering of the middle cards, while Player II does not. How would the optimal strategies of both players change to compensate for this inequality in knowledge, and would there be an appreciable increase in Player I's expected winnings? These are just a few of the possible directions in which future research on this topic could proceed.
BIBLIOGRAPHY


'Mathematical Reviews' code for this topic is 90D05.

Dewey Decimal code for this topic is QA269.
program goofspiel;

uses MemTypes, PasPrinter;

var
cardvalues = array[1..15] of integer;
matrixvalues = array[1..15] of extended;

function max (x, y : extended) : extended;
begin
  if (x >= y) then max := x else max := y;
end;

function min (x, y : extended) : extended;
begin
  if (x <= y) then min := x else min := y;
end;

{This function solves the 2x2 game exactly.}
function solve2 (a, b, c, d : integer) : extended;
begin
  if ((a >= b) and (c >= d)) then solve2 := max(b, d) else
  if ((b >= a) and (d >= c)) then solve2 := max(a, c) else
  if ((a >= c) and (b >= d)) then solve2 := min(a, b) else
  if ((c >= a) and (d >= b)) then solve2 := min(c, d) else
  solve2 := (a*d - b*c)/(a + d - b - c);
end;

{This function solves the 3x3 game by approximation. The variable Count determines the precision of the solution.}
function solve3 (a, b, c, d, e, f, g, h, i : extended) : extended;
begin
  temp1, temp2, temp3, lambda, minlambda : extended;
  x1, x2, x3, t1, t2, t3, count : integer;
begin
  writeln(a, ' ', b, ' ', c, ' ', d, ' ', e, ' ', f, ' ', g, ' ', h, ' ', i);
  count := 36;
  minlambda := -1000.0;
  for x1 := 0 to count do
    for x2 := 0 to (count-x1) do
      begin
        x3 := count - x1 - x2;
        temp1 := (a*x1)+(d*x2)+(g*x3);
        temp2 := (b*x1)+(e*x2)+(h*x3);
        temp3 := (c*x1)+(f*x2)+(i*x3);
        lambda := min(min(temp1, temp2), temp3);
        if lambda > minlambda then
          begin
            t1:=x1; t2:=x2; t3:=x3;
            minlambda := lambda;
            end;
      end;
  writeln(minlambda/count, ': ', t1/count, ': ', t2/count, ': ', t3/count);
end;
solve3 := minlambda/count;
end;

{This function solves the 4x4 game by approximation. The variable Count}
function solve4 (fmat : matrixvalues) : extended;
var
  temp1, temp2, temp3, temp4, lambda, minlambda : extended;
  x1, x2, x3, x4, t1, t2, t3, t4, count : integer;
begin
  begin
    printfa('a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i');
    count := 36;
    minlambda := -1000.0;
    for x1 := 0 to count do
      for x2 := 0 to (count-x1) do
        for x3 := 0 to (count-x1-x2) do
          begin
            x4 := count - x1 - x2 - x3;
            temp1 := (fmat[1]*x1)+(fmat[5]*x2)+(fmat[9]*x3)+(fmat[13]*x4);
            temp2 := (fmat[2]*x1)+(fmat[6]*x2)+(fmat[10]*x3)+(fmat[14]*x4);
            temp3 := (fmat[3]*x1)+(fmat[7]*x2)+(fmat[11]*x3)+(fmat[15]*x4);
            temp4 := (fmat[4]*x1)+(fmat[8]*x3)+(fmat[12]*x3)+(fmat[16]*x4);
            lambda := min(min(temp1, temp2, temp3, temp4));
            if lambda > minlambda then
              begin
                t1 := x1; t2 := x2; t3 := x3; t4 := x4;
                minlambda := lambda;
              end;
          end;
end;

function ind (e, f : extended) : integer;
begin
  if e > f then ind := 1
  else if e = f then ind := 0
  else ind := -1;
end;

{Given the remaining two cards in Player 1 and II's hands, and the middle,
this function returns the value of the payoff matrix.}

function twocard (a1,a2,b1,b2,fourmid,fivemid : integer) : extended;
var
  a,b,c,d : integer;
begin
  a := ind(cards[a1],cards[b1]) * cards[fourmid] + ind(cards[a2],cards[b2]) * cards[fivemid];
  b := ind(cards[a1],cards[b2]) * cards[fourmid] + ind(cards[a2],cards[b1]) * cards[fivemid];
  c := ind(cards[a2],cards[b1]) * cards[fourmid] + ind(cards[a1],cards[b2]) * cards[fivemid];
  d := ind(cards[a2],cards[b2]) * cards[fourmid] + ind(cards[a1],cards[b1]) * cards[fivemid];
  twocard := solve2(a,b,c,d);
end;

{This function somewhat laboriously figures out the elements for a 3x3
matrix and then returns the value of it.}

function nextplays (cards1,cards2,cardsmid : cardvalues; tmid : integer) : extended;
var
  u,v,w, x,y,z, total, fourmid,fivemid : integer;
  b,c,d,e,f,g,h,i : extended;
in
  cardsmid[tmid] := 1;
  if ((cards1[1] = 1) and (cards1[2] = 1)) then
    begin u := 3; v := 4; w := 5; end else
  if ((cards1[1] = 1) and (cards1[3] = 1)) then
    begin u := 2; v := 4; w := 5; end else
  if ((cards1[1] = 1) and (cards1[4] = 1)) then
if (cards[1] = 1) and (cards[4] = 1) then
  begin u := 2; v := 3; w := 5; end else
if (cards[1] = 1) and (cards[5] = 1) then
  begin u := 2; v := 3; w := 4; end else
if (cards[2] = 1) and (cards[3] = 1) then
  begin u := 1; v := 4; w := 5; end else
if (cards[2] = 1) and (cards[4] = 1) then
  begin u := 1; v := 3; w := 5; end else
if (cards[2] = 1) and (cards[5] = 1) then
  begin u := 1; v := 3; w := 4; end else
if (cards[3] = 1) and (cards[4] = 1) then
  begin u := 1; v := 2; w := 5; end else
if (cards[3] = 1) and (cards[5] = 1) then
  begin u := 1; v := 2; w := 4; end else
if (cards[4] = 1) and (cards[5] = 1) then
  begin u := 1; v := 2; w := 3; end;
if (cards[1] = 1) and (cards[2] = 1) then
  begin x := 3; y := 4; z := 5; end else
if (cards[1] = 1) and (cards[3] = 1) then
  begin x := 2; y := 4; z := 5; end else
if (cards[1] = 1) and (cards[4] = 1) then
  begin x := 2; y := 3; z := 5; end else
if (cards[1] = 1) and (cards[5] = 1) then
  begin x := 2; y := 3; z := 4; end else
if (cards[2] = 1) and (cards[3] = 1) then
  begin x := 1; y := 4; z := 5; end else
if (cards[2] = 1) and (cards[4] = 1) then
  begin x := 1; y := 3; z := 5; end else
if (cards[2] = 1) and (cards[5] = 1) then
  begin x := 1; y := 3; z := 4; end else
if (cards[3] = 1) and (cards[4] = 1) then
  begin x := 1; y := 2; z := 5; end else
if (cards[3] = 1) and (cards[5] = 1) then
  begin x := 1; y := 2; z := 4; end else
if (cards[4] = 1) and (cards[5] = 1) then
  begin x := 1; y := 2; z := 3; end;
if cardsmid[1] = 0 then begin fourmid := 1; cardsmid[1] := 1; end else
if cardsmid[2] = 0 then begin fourmid := 2; cardsmid[2] := 1; end else
if cardsmid[3] = 0 then begin fourmid := 3; cardsmid[3] := 1; end else
if cardsmid[4] = 0 then begin fourmid := 4; cardsmid[4] := 1; end;
if cardsmid[2] = 0 then fivevid := 2 else
if cardsmid[3] = 0 then fivevid := 3 else
if cardsmid[4] = 0 then fivevid := 4 else
if cardsmid[5] = 0 then fivevid := 5;
a := incards(u, cards[x]) * cards[mid] + twochar(v, w, y, z, fourmid, fivevid);
b := incards(u, cards[y]) * cards[mid] + twochar(v, w, x, z, fourmid, fivevid);
c := incards(u, cards[z]) * cards[mid] + twochar(v, w, x, y, fourmid, fivevid);
d := incards(u, cards[x]) * cards[mid] + twochar(v, w, y, z, fourmid, fivevid);
e := incards(u, cards[y]) * cards[mid] + twochar(v, w, x, y, fourmid, fivevid);
f := incards(u, cards[z]) * cards[mid] + twochar(u, w, y, z, fourmid, fivevid);
g := incards(u, cards[x]) * cards[mid] + twochar(u, v, y, z, fourmid, fivevid);
h := incards(u, cards[y]) * cards[mid] + twochar(u, v, x, y, fourmid, fivevid);
i := incards(u, cards[z]) * cards[mid] + twochar(u, v, x, z, fourmid, fivevid);

{ writeln(a:2:2, 'b:2:2, 'c:2:2); }
{ writeln(d:2:2, 'e:2:2, 'f:2:2); }
{ writeln(g:2:2, 'h:2:2, 'i:2:2); }
{ writeln('); }

nextplays := solve3(a, b, c, d, e, f, g, h, i);
end;

{This function sets up which cards have already been played from each}
function threecard(fp1,fp2,fmid,sp1,sp2,smid : integer) : extended;

uses
  cards1, cards2, cardsmid : cardvalues;
  thirdmid : integer;
  temp : extended;
begin
  cards1[fp1] := 1; cards1[sp1] := 1; cards2[fp2] := 1; cards2[sp2] := 1;
  cardsmid[fmid] := 1; cardsmid[smid] := 1;

  temp := 0;
  for thirdmid := 1 to 5 do
    if not(cardsmid[thirdmid] = 1) then
      temp := temp + nextplays(cards1,cards2,cardsmid,thirdmid);
  threecard := temp/3;
end;

{This procedure looks at all possible plays on the second turn. It writes the average value of the 4x4 matrices over the four possible next upturned middle cards.}

procedure fourcard(firstplay1,firstplay2,firstmid : integer);
var
  fourmatrix : matrixvalues;
  i,firstresult, seconddresult, secondplay1,secondplay2,secondmid, index : integer;
  temp : extended;
begin
  temp := 0.0;
  firstresult := index(cards[firstplay1],cards[firstplay2]) * cards[firstmid];
  for secondmid := 1 to 5 do
    if not(firstmid = secondmid) then
      begin
        index := 1;
        for secondplay1 := 1 to 5 do
          if not(secondplay1 = firstplay1) then
            begin
              for secondplay2 := 1 to 5 do
                if not(secondplay2 = firstplay2) then
                  begin
                    seconddresult := index(cards[secondplay1],cards[secondplay2]) * cards[secondmid];
                    fourmatrix[index] := (firstresult + seconddresult +
                                threecard(firstplay1,firstplay2,firstmid,secondplay1,secondplay2,secondmid));
                    write(printer,fourmatrix[index]:2:2,' ');
                    index := index + 1;
                  end;
                end;
          end;
        end;
      end;
    end;
  writeln(printer,' ');  
  writeln(printer,' '); temp := temp + solve4(fourmatrix);
  writeln(printer,' ');  
  writeln(printer,' ');  writeln(printer,(temp/4):3:3);
end;

{This procedure sets up the initial cards; both players and the middle are assumed to start with identical hands. The payoff to player 1, given by 'fourcard(a,b,c)', tells the expected payoff if on the first trick, Player 1 plays card a, Player 2 plays card b, and the middle card is c. In this particular example, each player and the middle start with cards valued at 2, 3, 5, 10, and 20. The program will return the expected
{payoff to Player 1 if, on the first turn, Player 1 discards a 3, Player 2 discards a 5, and the middle card is a 10.}

procedure driver;
  \[\text{in}\]
  \[\text{fourcard(2,3,4)};\]
  \[\text{readln;}\]
  \[\text{end;}\]

begin
  \[\text{driver;}\]
end.