1-1-2007

Estimating Winning Probabilities in Backgammon Races

Andrew M. Ross
Eastern Michigan University

Arthur T. Benjamin
Harvey Mudd College

Michael Munson '94
Harvey Mudd College

Estimating Winning Probabilities in Backgammon Races

Andrew M. Ross[1], Arthur Benjamin[2], and Michael Munson[3]

[1] Department of Mathematics, Eastern Michigan University
[2] Department of Mathematics, Harvey Mudd College
[3] Denali Advisors, LLC

Abstract. In modern backgammon, it is advantageous to know the chances each player has of winning, and to be able to compute the chances without the aid of calculators or pencil and paper. A simple model of backgammon is used to approximate those chances, and a readily computable and sufficiently accurate approximation of that is developed. From there, the model is compared to simulated backgammon games, and the previous approximation is modified to fit the real data.

1 Introduction

1.1 Motivation

Backgammon differs from other common games (checkers, chess, etc.) because there is a formal way to increase the stakes as the game progresses. It is important to know one's current chances of winning in order to decide whether or not to increase the stakes. This paper focuses on ways to estimate the current chances of winning using only mental calculations.

The doubling cube, which was invented in the 1920s, is how the stakes may be increased during the game. The cube starts in the middle of the board with a value of 1. When one player (call him A) decides that his chances of winning are just right, he gives the cube to his opponent (B) before A's turn to roll. The opponent can either accept or reject the cube. If B rejects, the game is over and he must pay up. If he accepts, the two players play for twice as much money as before. Then, B possesses the cube, and A cannot use it until A is doubled and accepts.

Incorrectly estimating the chances of winning can result in resigning too soon, wagering too much on a weak position, or not increasing the wager on a strong position. So, it is important to be able to accurately estimate the chances of winning, often to within 1 percentage point. We will abbreviate the chances of winning as WP, or the Winning Percentage or Probability.

However we decide to estimate the WP, it must be easy enough for a human to do it in a minute or two. Players are not allowed to use computers, calculators, printed tables, or pencil and paper when playing a game.

Academic research on backgammon has been largely confined to two areas: strategies for using the doubling cube (offering or accepting), and computerized backgammon players. Examples of the former include Thorp (1975; 2007, this volume, pp. 237–265), Keeler and Spencer (1975), Orth (1976), Zadeh and Kobliska (1977), Zadeh (1977), and Buro (1999); these generally either assume that the WP is known before making any decision about doubling, or they discuss ways to compute the WP that are not suitable to mental calculation. Thorp (1975, 2007) does discuss some mental calculations, but those results were not available to the present authors until recently. Computerized players, often based on neural networks, are now better than the best human players (Tesauro 2002).

1.2 The game of backgammon

Backgammon is played on a board with 15 checkers per player, and 24 points (organized into four tables of six points each) that checkers may rest on. Players take turns rolling the two dice to move checkers toward their respective "home boards," and ultimately off the board. The first player to move all of his checkers off the board wins.

When the dice are rolled, they are applied sequentially to the board by moving one checker by the amount on one die, and then another checker by the amount on the other die. The checker moved with the second die may be the same one that was moved with the first die. Thus, if a player has only one checker on the 3 point and another on the 6, and rolls a 2 and a 1, then the checker on the 6 point may be moved to the 4 point, and the checker on the 3 point may be moved to the 2 point (written 6-4 3-2). Alternatively, the player may play 6-5 3-1 or 6-5 5-3. If doubles are rolled, the player may move four checkers. Thus, if double-ones were rolled in the above situation, the player could move 6-5 5-4 3-2 2-1.

Because of the rule for doubles, the mean of the dice rolls in backgammon is $\mu = 49/6 \approx 8.1667$, and the standard deviation is $\sigma = \sqrt{665/36} \approx 4.3$. No moving checker may land on a point that has two or more opponent checkers. If a point has exactly one opponent checker (a "blot"), it may be landed on and "hit." This complicates the game, so all movements mentioned in this paper will strictly avoid landing on opponent's checkers.

After the two players can no longer hit each other, they enter a "race" to get their checkers off the board. First, they "bear in" checkers onto their 6 home board points. Once all 15 of a player's checkers are in, that player begins the "bearoff," moving checkers off the board. This is done by moving checkers off the points that correspond to the dice rolls. If no checker appears on that point, one from a higher point must be moved down. If there are no checkers on higher points, the highest checker may be borne off.

This paper will focus on the race.

1.3 Approaches to estimating the WP

There are approximately 300 trillion possible racing games, when both players' positions are considered. Our approach to estimating WPs is to develop a way to "measure" a position to get an idea of how good it is. From the measurements of the two players' positions, we estimate the WP by computing a function or looking something up in a (memorized) table. When we compute WP values, we use only the behavior of the dice and checkers; we do not include the possibility of winning or losing when a double is offered or declined. If we name the player on roll A, and the opponent B, we will have the system for estimating a WP shown in Figure 1 Once we decide how to measure a

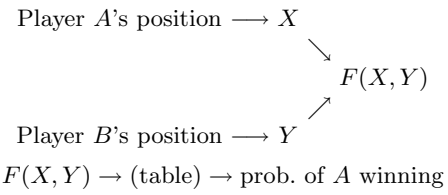

Fig. 1. System for estimating WP.

position, we must figure out an easily computable function $F(X, Y)$ which gives a good estimate of the WP.

The most common way to measure a position is to compute the Raw Pip Count, or RPC. This is the sum of the distance that each checker has to bear off. Thus, if a player has two checkers on the three point, and one checker on the four point (and no other checkers on the board), the RPC is 10. RPC is an integer value that may be quickly computed and kept track of while playing an actual game, so it is of use as a measurement of position. If two players have the same RPC, they have roughly the same chance of winning the game, with the player on roll having a small advantage. At the start of the game, the RPCs are 167 each. At the start of the race, they are usually 80 to 100 each.

It is entirely possible for two different positions to have identical RPCs. Therefore, if we use only the RPC to measure a position, we lose some information that might have an effect on the WP. Later, we will estimate this loss by testing different positions that have the same RPC.

In order to know what the function $F(X, Y)$ should be, we must know the actual WPs. We estimate these by simulating many, many games and keeping track of who wins each one. This process is known as "rolling out" a position. Once this is done, we are left with the task of inventing an easy-to-compute function that matches the data to within 1 percentage point.

To aid in finding such a function, we will investigate a simpler version of backgammon called the "single checker model," or SCM. It give us WPs without the need to play many games. We will then compare this simpler version to the rollout data to see how well the two correspond.

Our paper is organized as follows: In Section 2 we introduce the SCM, and compute its exact WPs. We then discuss an existing approximation and its variations. In Section 3 we discuss how we set up our rollout (simulation) experiments, how the WP results compare to the SCM, and how to adjust an approximation method to better correspond to our results.

2 The single checker model

The SCM assumes that a position with an RPC of X is equivalent to having a single checker X pips away from bearing off. This is easiest to see for very low pip counts. For example, if there are two checkers on the one-point and one on the two-point, the SCM considers one checker on the four-point as an equivalent position. This gets more complicated for larger X, since there are only 24 pips on the backgammon board. For that reason, it is more convenient to imagine the SCM as a discrete linear racetrack, with the opponents running in different lanes, neither able to influence the other.

In the SCM, players roll the dice and move their checkers exactly the number of pips shown on the dice (taking into account the doubles rule). The backgammon rule about bearing in before bearing off does not apply. The rule about using large dice rolls to bear off from lower-numbered points also does not apply. The first player to land on or overshoot zero wins.

Since the players alternate rolls, the player on roll has an advantage. He could bear off before the other player gets to roll. The player on roll is not necessarily the player who rolled first in the game. Rather, the advantage switches back and forth at each roll. We can put an exact value on this advantage. It is one-half of the expected value of a roll μ, or $4\frac{1}{12}$ pips. So, if player A has a raw pip count of X, and player B has a raw pip count of $Y = X - 4$, the two are approximately evenly matched.

Any constant difference between the two players becomes less and less important as the race gets longer. If player A has an RPC of 10 and player B has an RPC of 30, then player B is not likely to win. But, if the two RPCs are 130 and 150, respectively, then the two are more evenly matched. We expect to see each of the above-mentioned properties when we actually calculate the WPs.

2.1 Computing the WPs for the SCM

We will compute the WPs from the perspective of player A, the player on roll. These calculations were first published by Zadeh and Kobliska (1977), though they had also been presented by Thorp (1975). Let $\mathrm{WP}(X, Y)$ be the WP

when the player on roll, A, has X pips to go, and B has Y pips to go. Denote the probability of rolling n pips as $p(n)$. Immediately after the roll, the two players swap situations, but player A has moved some number of pips. So, we sum over all possible rolls:

$$\mathrm{WP}(X, Y) = 1 - \sum_{n=3}^{24} p(n)\,\mathrm{WP}(Y, X - n). \tag{1}$$

We must be careful near zero. We define $\mathrm{WP}(X, Y) = 1$ for $X \leq 0$ and $Y > 0$, and $\mathrm{WP}(X, Y) = 0$ for $X > 0$ and $Y \leq 0$.

This is like a dynamic programming problem. The WP values can be computed by simple recursion or iteration. A simple program takes no more than a minute to compute WP for values of X and Y between 0 and 200. The surface that this produces has the properties predicted for it. Its 3-dimensional shape is not terribly interesting, but we will need to look at values from it in a graphical manner. We will do this by taking cross-sections parallel to the Y-axis (constant X value). Another way to think about it is that the X axis comes "out of the page" toward the reader.

Figure 2 shows some of these cross-sections. Here we can see the properties

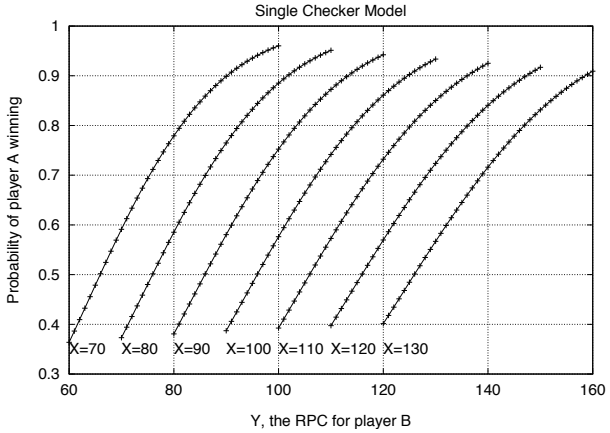

Fig. 2. Single checker model predictions for X in increments of 10.

mentioned above. The 50% point is not at the point where $Y = X$; instead, it is near $Y = X - 4$. It is not exactly there, since the value of being ahead is not exactly 4. Also, we see that a constant difference has less of an effect as both RPCs grow. For example, the points where $Y = X + 10$ tend downward.

2.2 Approximating the SCM

Although the SCM is substantially simpler than real backgammon, it is still impractical to memorize the large table $\mathrm{WP}(X, Y)$ computed above. We need an easily computable function that gives us the values we need.

We might start by looking for simple functions $F(X, Y)$ that will lead to a simple table lookup. The simplest reasonable function would be $F(X, Y) = Y - X$, the lead that player B has. However, there are a wide range of WP values for any particular player B lead. Other simple functions would be $F(X, Y) = (Y - X)/X$ or $(Y - X)/Y$, the player B lead proportional to the current position. These also lead to a wide range of WP values at any particular F value, so they are not too useful either.

A much better solution, based on renewal theory, has been proposed by Kleinman (1980). It is called the D^2/S system. We consider the SCM as a renewal process, where each roll of the dice corresponds to an item lifetime. We need to know how many rolls are needed to bear off, which corresponds to how many items are needed in a particular length of time in a renewal process. We will use the Central Limit Theorem for Renewal Processes (CLTRP) from Feller (1968), which we paraphrase here:

> If independent random variables with mean μ and variance σ^2 are to be summed until some large number T is reached, the number of variables needed is approximately normally distributed with mean T/μ and variance $\sigma^2 T/\mu^3$.

This approximation merely means that the ratio of the true mean to T/μ goes to 1 as T grows, and the ratio of the true variance to $\sigma^2 T/\mu^3$ also goes to 1 as T grows. Applying this to backgammon, T is the total number of pips a player has left to move, and the random variables to be summed are dice rolls. The "number of variables needed" is the number of rolls needed to bear off.

One important question is whether the RPCs common in real backgammon are large enough that the normal approximation applies. Another question is what the actual mean and variance are. For now, we will assume that the approximation applies, and we will use the mean and variance of the number of rolls as given by the theorem.

Let n_X be the number of rolls to bear off a single checker at X points from the origin, and similarly for n_Y and Y. According to the theorem, we have

$$n_X \sim N\left(\frac{1}{\mu}X, \frac{\sigma^2}{\mu^3}X\right) \tag{2}$$

and

$$n_Y \sim N\left(\frac{1}{\mu}Y, \frac{\sigma^2}{\mu^3}Y\right). \tag{3}$$

The chance that the player on roll wins is the probability that $n_X \leq n_Y$, or $n_Y - n_X \geq 0$, where

$$n_Y - n_X \sim N\left(\frac{1}{\mu}(Y-X), \frac{\sigma^2}{\mu^3}(Y+X)\right). \tag{4}$$

We then define Z to be a standard normal variable:

$$Z \equiv \left(n_Y - n_X - \frac{Y-X}{\mu}\right)\sqrt{\frac{\mu^3}{\sigma^2(Y+X)}} \sim N(0,1). \tag{5}$$

At this point, we must stop and recall that we are approximating a discrete distribution with a continuous one. When we consider $\mathrm{P}(n_Y - n_X \geq 0)$, we will approximate a finite sum with an infinite integral. So, we must perform a continuity correction. This is usually done by integrating to a point halfway between two of the integer points. In this case, we want to include 0 in the integral, and we are integrating toward positive infinity. Therefore, we will start at -0.5.

Given this standard normal distribution, we may calculate the probability of the player on roll winning:

$$\begin{aligned}\mathrm{P}(n_Y - n_X \geq -0.5) &= \mathrm{P}\left(Z \geq \left(-\frac{1}{2} - \frac{Y-X}{\mu}\right)\sqrt{\frac{\mu^3}{\sigma^2(Y+X)}}\right) \\ &= \Phi\left(\frac{Y-X+\mu/2}{\sqrt{Y+X}}\frac{\sqrt{\mu}}{\sigma}\right),\end{aligned} \tag{6}$$

where $\Phi()$ is the Cumulative Distribution Function (CDF) of the standard normal. Now we define $D \equiv Y - (X - \mu/2)$ and $S \equiv Y + X$. Note that D has an extra term in it, while S does not. This extra term results from the continuity correction, but it also can be interpreted as adjusting for the $\mu/2$ advantage that the player on roll has. Simplifying the above with this expression, we get

$$\Phi\left(\frac{D}{\sqrt{S}}\frac{\sqrt{\mu}}{\sigma}\right). \tag{7}$$

By this, we see that if A is behind in the race ($X - \mu/2 > Y$), D will be negative and A's probability of winning will be low, as expected.

How does the formula in (7) become D^2/S? The way most people would compute that formula is by computing its argument, and then looking up the result in a memorized table. But, extracting square roots is not an easy mental calculation. So, we square the whole argument to get

$$\frac{D^2}{S}\frac{\mu}{\sigma^2} \tag{8}$$

and adjust the table accordingly. From there, divide the constant out and again adjust the table. These adjustments are fairly transparent to those who want to use the D^2/S method, since it does not really matter what numbers must be memorized.

Kleinman makes an adjustment to the constant μ/σ^2 based on an observation about dice rolls. We will investigate this below.

2.3 Improving the approximation

Kleinman's D^2/S system provides a fairly good approximation to winning probabilities in the SCM. However, it can be improved using more advanced results from renewal theory. The CLTRP is true only in the asymptotic sense. That is, it actually says that

$$\lim_{T\to\infty} \frac{\mathrm{E}[\#\text{ of rolls required}]}{T/\mu} \to 1 \quad \text{and} \quad \lim_{T\to\infty} \frac{\mathrm{Var}(\#\text{ of rolls required})}{T\sigma^2/\mu^3} \to 1. \tag{9}$$

There are actually constant offsets for the mean and the variance. The offset for the mean is:

$$\lim_{T\to\infty} \mathrm{E}[\text{number of rolls required}] - T/\mu = \frac{1}{2}\left(\frac{\sigma^2}{\mu^2} - 1\right), \tag{10}$$

and the offset for the variance is fairly complicated (it is given by Tijms 1994). Substituting our values for μ and σ, we get an offset of -0.3615 (rounded). This then needs two adjustments: Ordinary renewal theory does not count a renewal at time zero, while we count it as the first roll, so we add 1 to compensate, getting 0.6384. Also, ordinary renewal theory is in continuous time, while our system is discrete, so we subtract 1/2 pip as a continuity correction. This means subtracting $(1/2)/\mu = 0.061224$ rolls, for a final offset of $198/343 \approx 0.5772$, which we will define as δ_μ. This value has been verified by formulating a discrete-time Markov chain that models the current RPC and how it decreases with the dice rolls. The Markov chain is detailed below. To find the variance offset, it was simpler to use the Markov chain's results than to apply continuity corrections to the usual renewal process variance offset. We ultimately found a variance offset of $\delta_{\sigma^2} \equiv -0.0277945$.

Now we know what the asymptotic values are for the mean and variance of n_X:

$$\mathrm{E}\left[n_X\right] = \frac{1}{\mu}X + \delta_\mu, \tag{11}$$

$$\mathrm{Var}\left[n_X\right] = \frac{\sigma^2}{\mu^3}X + \delta_{\sigma^2}. \tag{12}$$

Using the same logic as before, we end up with

$$\Phi\left(\frac{D}{\sqrt{\sigma^2 S/\mu + 2\mu^2\delta_{\sigma^2}}}\right). \tag{13}$$

Notice that since each player has a δ_μ term, they cancel each other. If we apply this adjustment, we see (Figure 3) that the error in the approximation is reduced by a factor of 2 or so.

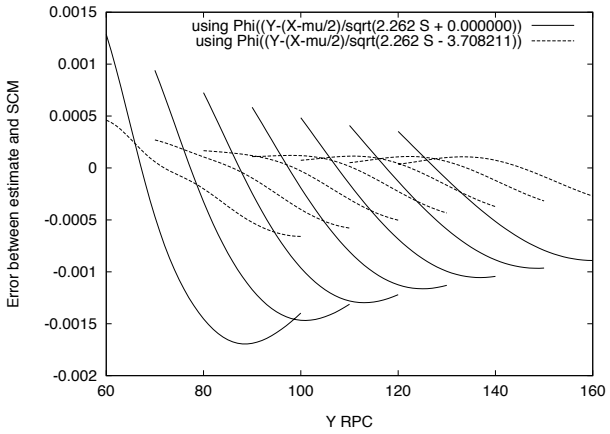

Fig. 3. Adjusting the D^2/S method.

2.4 The Markov chain SCM

As mentioned above, we formulated a discrete-time Markov chain based on the SCM to verify the mean offset δ_μ and compute the variance offset δ_{σ^2}. Next, we describe that Markov model.

For each RPC in the model, we will have a state in the Markov chain. Taking 160 as the upper bound on races for now, we will have the 161 Markov states $0, 1, \ldots, 160$. The chances of moving from state i to state j in one roll will be determined by what is allowed by the dice. Figure 4 shows how this works when moving from state 160; the same graph applies for all the states above 24. The thickness of each arrow in the figure shows how probable that transition is. Near zero this graph must change, since overrunning zero is as

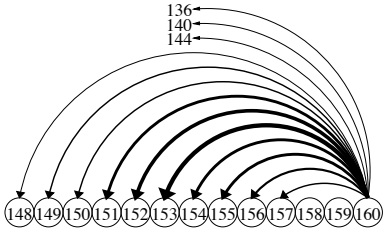

Fig. 4. How to form the bear-in portion of the Markov matrix.

good as landing on it directly (they both end the game). Figure 5 shows how some of the arrows are re-routed. Once we hit zero, we move directly to a "done" state, where we stay. This is known as an absorbing state, and it helps keep track of exactly when we finished.

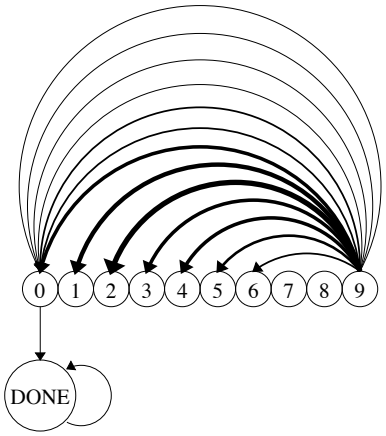

Fig. 5. How to form the bear-off portion of the Markov matrix.

Our matrix M has some nice properties. One is that $M(i,i) = 0$ (except for the Done state), since we can't stay in place when we roll the dice. Also, $M(i,j) = 0$ for $i < j$, since we can't move backwards. So, we have a lower-triangular matrix (with the exception of the Done state). For any row $i \geq 24$, we have $M(i,j) = p(i-j)$, as shown in Figure 4. For the rows $i < 24$, we have

$$M(i,0) = \sum_{j=i}^{24} p(j). \tag{14}$$

From the above description, we compile our matrix M. Its first row and column correspond to the "done" state. Its second row and column correspond to the 0 RPC state; its third row and column correspond to an RPC of 1, etc. With the one exception of the upper left-hand corner, this matrix is strictly lower triangular. This matrix can be expressed schematically as in Figure 6. The trapezoid represents the RPCs that cannot bear off in one turn. The triangle represents those RPCs near zero where special summing must be done. The rectangular block shows entries that take care of moving to the "done" state.

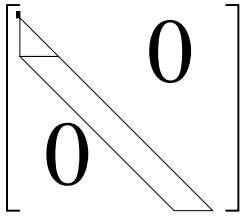

Fig. 6. A schematic view of the Markov matrix.

Once the matrix is formed, we can compute the mean and variance of the time to finish in two ways. One way is to compute the full distribution of the time to finish by raising the matrix to higher and higher powers: the probability of moving from state i to state j in exactly k turns is the (i, j) component of M^k. So, to figure out the distribution of the number of turns to bear off from X pips, we use

$$\mathrm{P}(n_X = k) = M^k(X, 0). \tag{15}$$

The mean and variance are also available without computing the full distribution, using results from the theory of first passage times. Now that we have a more full understanding of how to adjust the D^2/S method using mean and variance offsets from the SCM, we turn to more-realistic simulations of racing games.

3 The differences between the SCM and real backgammon

As we have seen, the D^2/S model is a very good over-the-board approximation to the SCM. The question remains, though: how good is the SCM as an approximation to real backgammon races? Real backgammon has rules about bearing off that often force players to "waste" pips on the dice. That is, quite often a roll of 5 (for example) will be used to bear off a checker from the 3-point, wasting 2 pips. The closer the game gets to the end, the more often pips are wasted, since the checkers are on the low-numbered points. The SCM takes into account only one way wastage can happen: on the very last move. In reality, when the RPC is at (say) 30, the possible moves are restricted by the bearoff rules. Since the RPC is over 24, though, the SCM neglects the possibility of wastage.

Real races can also stretch the rule about no contact between the players. For pip counts above 85 or so, it is common for pieces to remain on one's 13-point. If this is the case for both players, contact has not truly been avoided. However, it is often the case that both players can avoid contact by avoiding the other's pieces and not leaving blots. This is different from the SCM, since the players must avoid landing on the others' pieces.

To determine how different real backgammon is from the SCM, we will simulate backgammon games, and keep track of the WP for each RPC. To do this, we created plausible initial positions for each RPC, and an easily implemented strategy for the computer to follow in moving checkers during the race.

3.1 Initial positions

There are many possible backgammon positions for each RPC that we are interested in. We could get an RPC of 90 by stacking all 15 checkers on the

6-point, but such a position would not happen very often in practice. We want positions that satisfy the following criteria:

- Realistic home board positions,
- Realistic outer board positions,
- Realistic numbers of crossovers (moving from one board to the next),
- Won't interfere with each other, and
- Two positions with close RPCs are only different by a few checkers.

By "interfere," we mean that if we try to match two of these positions against each other, they won't call for both players to occupy any one point. That way, we can use only one position for each RPC, and then generate games for each RPC pair by just using those two positions. The last constraint means that, ideally, we will change from 90 to 91 raw pips by moving one checker by one pip, instead of shuffling around multiple checkers. This will not always be possible, since we will have to jump over opponent's pieces on our 11 and 12 points (the noninterference constraint).

Positions matching these constraints were generated. Figure 7 shows the 70 position matched against the 90 position, and Figure 8 shows the 130 position matched against the 160 position. To test the sensitivity of the rollouts to these initial positions, other sets of positions were generated. One set had an extra crossover in each position, another set had two extra crossovers, and a third had a gap (no checkers) on the 5 point.

When we propose a strategy for the computer to use, the question of optimality immediately arises. How do we know that this is the best strategy? We don't. However, the strategy proposed below is good enough that real players use it, and we are interested in predicting results for games played by people. Also, both computerized players will use exactly the same strategy, so the results will not be biased. In the future, we might change the strategies slightly and have a small set of them to play against each other.

Bearing in

Our strategy for bearing in is based on the principle of doing crossovers (moving a checker from one table to another) as soon as possible. It consists of two components, called H and I. Component H considers the larger die first. It looks for a checker which crosses over a table boundary and is the farthest one from home that does so. If it finds such a checker, it moves it, then tries to do the same with the smaller die. Of course, if doubles were rolled, the sizes of the rolls are irrelevant, and H goes through its scanning four times.

If H left more than one die to use, the remaining dice are summed, and H is run again. When all ways of calling H have failed (no more checkers can cross over with the dice remaining), I is called. It looks for the farthest checker back that lands on a point not occupied by any other pieces, whether those pieces belong to the moving player or the opponent. This will avoid stacking

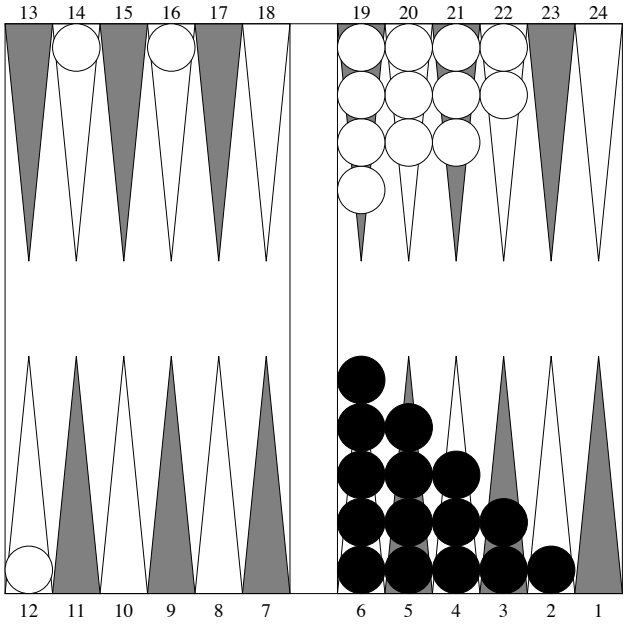

Fig. 7. 70 (black) versus 90 (white).

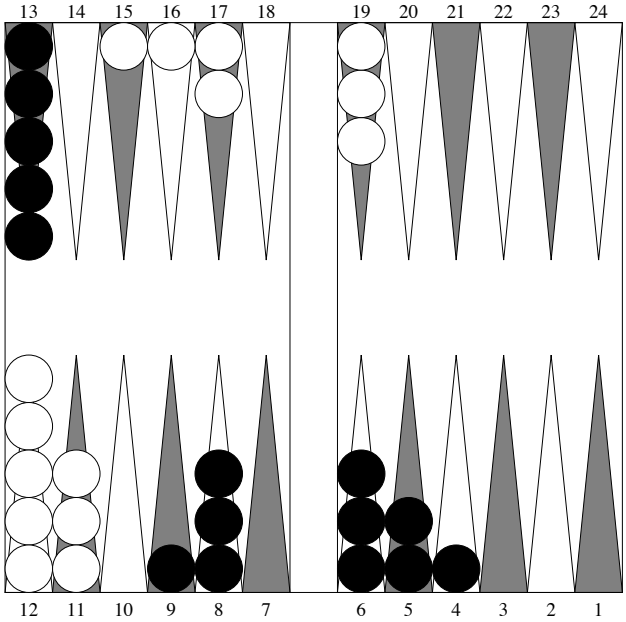

Fig. 8. 130 (black) versus 160 (white).

a player's checkers before bearoff. Failing that, I looks for the farthest checker back that doesn't land on an opponent.

Once all pieces are in the home board, we switch strategies entirely.

Bearing off

Our bearoff strategy will be based on a measurement of the position called the Expected Pip Count, or EPC. We will try to minimize the EPC at each move. This is almost always a good strategy. Buro (1999) mentions that it can be suboptimal when the game is almost over (low RPC values), but overall it is reasonable. Here is how this strategy works.

As play progresses in the bearoff, the player will roll more pips than are needed to bear off completely. For example, starting bearoff with an RPC of 70, a player might roll a total of 90 pips on the dice before being completely done. The number of pips actually rolled is a random variable. It must be at least equal to the RPC, and there is an upper limit on how large it can be, since racing bearoffs are guaranteed to finish in finite time. Since the range of this random variable is limited, it has a mean, or expected, value. We will call this the expected pip count, or EPC. Since we know the expected number of pips per turn, μ, we can calculate the expected number of turns until completion by simply dividing the EPC by μ. If we neglect the opponent's position, we will always favor a position with a low expected number of turns to a position with a higher expected number of turns. Since expected number of turns and EPC are linearly related, the EPC is a measure of how good a position is. This leads to a simple strategy:

1. Roll the dice.
2. Compute all positions we could move to.
3. Compute the EPCs for all of them.
4. Move to the one with the lowest EPC.

The hardest part of this is calculating the EPC for a position. For a very few simple positions, the EPC is easy to compute because the expected number of turns is either zero or one. The only position with an expected number of turns of zero is when all checkers are off the board. The positions where one is guaranteed to bear off in one turn are: one checker on the ace point, one checker on the two point, one checker on the three point, two checkers on the ace point, and one checker on each of the ace and two points.

¿From these six positions, we can calculate the EPC for any bearoff position. The method we will use is dynamic programming. For any position, we consider each possible roll of the dice in turn. For each roll, we generate all legal moves. For each of these new positions, we figure out the EPC. We then pick the move with the lowest resulting EPC. Once we have these minimum EPCs for all possible rolls, we average them. This is the average EPC once we move from this position. To move from this position, though, takes precisely one turn, or μ expected pips. This is expressed in Figure 9. Of course, a roll

$$\text{EPC for position } P = \begin{array}{llll} (\text{best EPC if a} & \text{1-1} & \text{is rolled} & + \\ \text{best EPC if a} & \text{1-2} & \text{is rolled} & + \\ & \vdots & & \\ & \vdots & & \\ \text{best EPC if a} & \text{6-6} & \text{is rolled)}/36 & + \\ \mu & & & \end{array}$$

Fig. 9. Calculations for EPC.

of 1-2 is the same as 2-1, and likewise for any nondoubles roll, so we compute those moves only once, and then weight them twice as much as the doubles rolls.

How many possible single-player bearoff positions are there? There are at most 15 checkers to distribute among 6 points; if K_i denotes the number of checkers on the i-th point, then it must be the case that $K_1 + K_2 + \cdots + K_6 \leq 15$. If we add a "dummy" point, point 0, that counts the number of checkers off the board, we have $K_0 + K_1 + \cdots + K_6 = 15$. The number of ways to do this is (from Feller 1968)

$$\binom{7+15-1}{15} = \frac{(7+15-1)!}{15!(7-1)!} = 54{,}264. \tag{16}$$

To store the EPC for each bearoff position, we need an array of size 54,264 and a method to translate from a backgammon position into an array position (an integer) and back. An algorithm or function that reduces a relatively complex situation (e.g., a backgammon position) into an integer is called a hash. If it is an injective function (so that no two positions map to the same integer), it is called a perfect hash (Cormen, Leiserson, Rivest, and Stein 2001, Section 11.5). As an example, taking the RPC of a position is a hash but it is not perfect, because many different positions can map to the same RPC. A perfect hash was developed by Benjamin and Ross (1997) to map a single player's bearoff position (number of checkers on each point) into an integer between 0 and 54,263, along with a reverse-hash to convert integers into racing positions. This hash is also minimal, in that none of the integers in the range are "wasted"—each integer maps back into a meaningful bearoff position. This hash is also extendable to more board points than just the 6 points of the home board.

This enumeration makes calculating EPCs fairly easy. We create an array of 54,264 real numbers, and initialize the locations corresponding to the six positions for which we know the EPCs. Then, we march up the integers, generating rolls and moves for each position and storing the EPCs as we calculate them. Conveniently, each legal move from the current position produces a position with a smaller integer representation, so we've already computed the EPC for it. Therefore, we don't have to call the EPC-calculating program

recursively, since we are being careful about the order in which we calculate positions.

At this point, we have EPCs for each of the 54,264 bearoff positions. This makes brute-force analysis of certain moves feasible. For example, consider the hypothesis "The best possible move is one that bears off two or four checkers (depending on doubles) if it possible to do so." This could be called a greedy algorithm, since it concentrates on getting checkers off the board at the expense of having a smooth home board. The question is, is such smoothing overrated? It turns out that the greedy approach is not always the best, but it is not a bad alternative. There are 1668 pairs of position and dice roll that result in the greedy choice being sub-optimal. However, the worst one differs by only 1.35 expected pips, which is about one-sixth of an expected turn. The position with this difference is position 41258, which is, in vector form, $(0, 5, 1, 8, 1, 0, 0)$, with a roll of 2-4. This position is obviously in need of some smoothing. The greedy move would leave five checkers on the 1-point and eight on the 3-point. The optimal move is 4-2, 3-off. This helps fills a gap on the 2-point. However, the greedy and optimal strategies do not produce results that are drastically different, so it might be easier to just go with the "2-or-4" strategy over the board.

Another interesting hypothesis is that it is always best (in terms of EPC) to use an ace to bear off from the 1-point if it is possible. An exhaustive search shows this is true.

3.2 One-player rollouts

Now that we have a strategy for moving checkers, we can compare the SCM to real backgammon. We will start by using the one-player versions of each, to see if the number of turns is still approximately normal. This one player will move simulated backgammon pieces (as opposed to the SCM), but the opponent's checkers will not be on the 12 and 11 points. So, the main difference between the SCM and the single-player rollout is the bearoff rules.

In Figure 10, we plot the mean number of turns from the rollout data and from the unadjusted CLTRP. On the left edge of the graph, the difference is roughly 1.1 raw pips, and on the right edge it is 1.0 raw pips. However, our calculations above show that the differences in the means end up canceling, so we will not investigate it further.

Next, we look at the variance of the number of turns to finish. Above, we mentioned that Kleinman has proposed a modification of the variation term in some of the formulas. His suggestion is that, in real games, the standard deviation of the dice rolls is closer to 4 than 4.3. Figure 11 shows that, indeed, the rollout variance systematically differs from what the CLTRP predicts for the SCM. It is closer to what Kleinman said. However, it is lower for the low RPCs, and higher for large RPCs. A best-fit line is also plotted, showing that the slope of the observed line is closer to the CLTRP value than to Kleinman's line, but the line is translated by roughly 13 pips to the right.

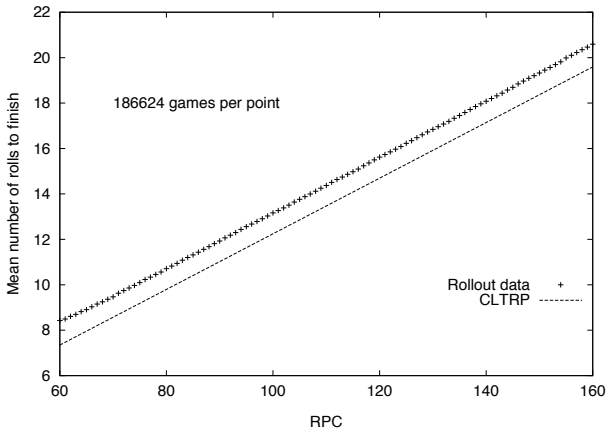

Fig. 10. The observed mean in a single-player rollout exceeds the CLTRP value.

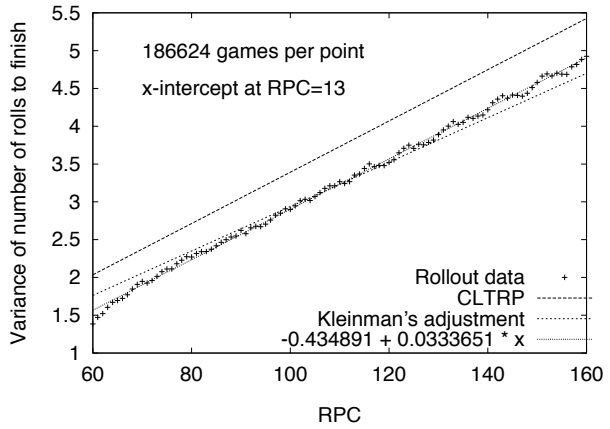

Fig. 11. Linear approximations to the observed variance in a single-player rollout.

3.3 Two-player rollouts

Having looked at how well the SCM matches ordinary single-player rollouts in terms of mean and variance of number of rolls, we next turn to our overall goal of calculating WPs in two-player games. For each position, we aim for a standard error in our WP value of 0.001, or 0.1%. This requires at least 250,000 games per initial position pair when WP is near 1/2.

Rather than running all of our created positions against each other, we focus on stepping X in increments of 10 from 60 to 130, and stepping Y from $X - 10$ to $X + 30$ in increments of 1. Figure 12 shows how our rollout data compare to the SCM predictions. We see that the SCM underpredicts the

WP values when they are above roughly 1/2, and overpredicts when the WP values are lower. Below, we will adjust the D^2/S model for this fact.

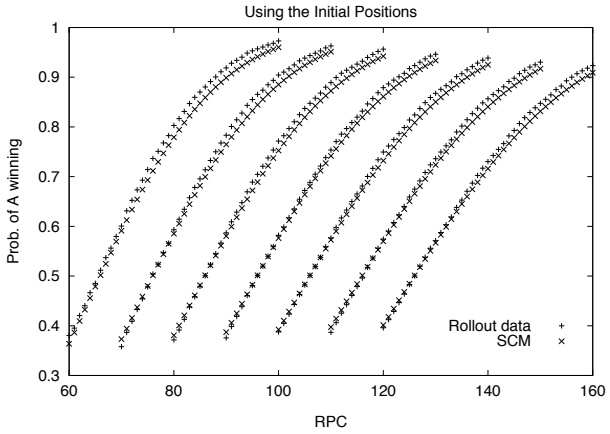

Fig. 12. Comparing the rollout data to the SCM.

Another potentially important factor is gaps in the initial boards. The initial positions were altered to produce large gaps. The resulting positions were then played against the original positions. For each (X, Y) pair, four sets of trials were run: no gap vs. no gap, gap vs. no gap, no gap vs. gap, and gap vs. gap. The results were not much different from the original positions. Figure 13 shows the effect of both players having a gap. It appears that the WP values are systematically higher (above 1/2) when both players have a gap for the first 5 curves, but lower for the last two. Figure 14 shows that larger differences occur when only one player has a gap. Still, the differences are not all that large.

Adjusting D^2/S for 2-player games

To adjust D^2/S to match our WP figures above, and still keep it reasonable to compute mentally, we start by looking at the observed mean and variance of number of turns to bear off from our two-player rollout data. Using a least-squares line fit, we find

$$\mathrm{E}\left[n_X\right] = 0.90544 + 0.122688X, \tag{17}$$

$$\mathrm{Var}\left[n_X\right] = -0.408285 + 0.0330249X, \tag{18}$$

with correlation coefficients of 0.999959 and 0.998989, respectively. When we put these into the formulae used in deriving D^2/S, we will use u for the slope of the mean, and v for the slope of the variance. Using the same logic as twice before, we again see that the δ_μ terms cancel each other, and we end up with

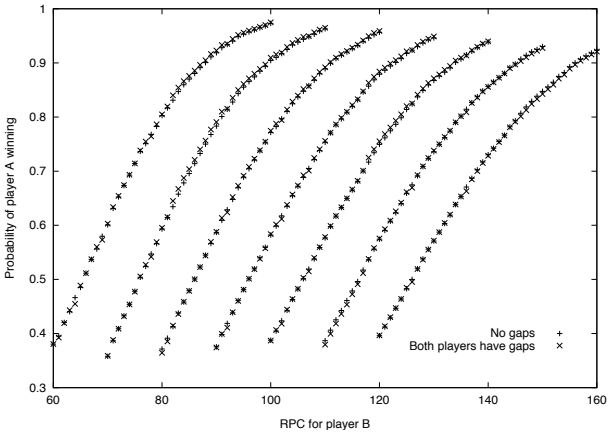

Fig. 13. Both players have a gap in their initial positions.

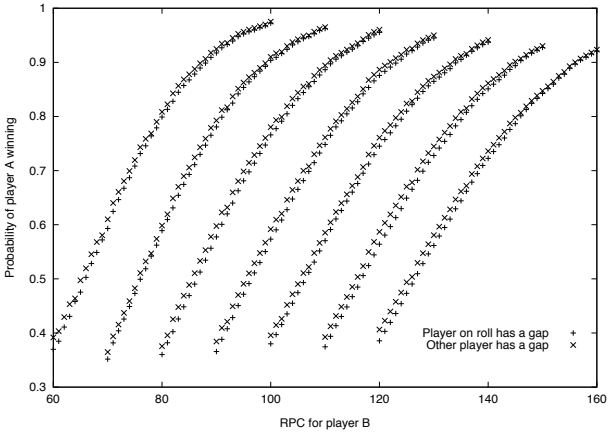

Fig. 14. Only one initial position has a gap.

$$\Phi\left(\left(\frac{1}{2}+u(Y-X)\right)\frac{1}{\sqrt{v(Y+X)+2\delta_{\sigma^2}}}\right). \tag{19}$$

We can transform this to

$$\Phi\left(\left(Y-(X-\frac{1}{2u})\right)\frac{1}{\sqrt{v(Y+X)/u^2+2\delta_{\sigma^2}/u^2}}\right) \tag{20}$$

to make it easier to compute. Again to make things easier later, we define ϵ as follows:

$$\frac{1}{2u} \approx 4.0753 \equiv 4+\epsilon. \tag{21}$$

Now we square the argument of Φ above, to avoid taking the square root:

$$\frac{\left(Y-(X-\frac{1}{2u})\right)^2}{v(Y+X)/u^2+2\delta_{\sigma^2}/u^2}. \tag{22}$$

Now multiply by u^2/v to eliminate the multiplication on the bottom:

$$\frac{\left(Y-(X-\frac{1}{2u})\right)^2}{(Y+X)+2\delta_{\sigma^2}/v}. \tag{23}$$

Now we will use ϵ to make the top a little bit more accurate. Usually, we would round $1/(2u)$ to just plain 4, but if we expand it beforehand, we can get a slightly better estimate:

$$\begin{aligned}\left(Y-\left(X-\frac{1}{2u}\right)\right)^2 &= (Y-(X-4)+\epsilon)^2 \\ &= (Y-(X-4))^2+2\epsilon(Y-(X-4))+\epsilon^2. \end{aligned}\tag{24}$$

We will ignore the ϵ^2, because it is approximately 0.005. However, $2\epsilon \approx 1/7$, which can amount to 5 pips in some races.

In the denominator, $2\delta_{\sigma^2}/v \approx -24.72588$. We will approximate this with -25, since there is little else that can be done to make it easy to calculate. Thus, our final computation is

$$\begin{array}{rl} \text{Player } A\text{'s position} \xrightarrow{\text{RPC}} X & \searrow \\ & \varDelta \equiv Y-(X-4) \\ & S \equiv Y+X \\ \text{Player } B\text{'s position} \xrightarrow{\text{RPC}} Y & \nearrow \end{array} \tag{25}$$

$$\frac{\varDelta^2+\varDelta/7}{S-25} \rightarrow \text{Table 1} \rightarrow \varPi. \tag{26}$$

Now if the player on roll is behind and $\varPi$ is above 50%, subtract $\varPi$ from 100% to get the proper probability of winning. Our table gives much more precision than would be used mentally, in case it is useful in future studies.

If the numerator is rounded to an integer, and -25 is used in the denominator instead of -24.7, an error of up to 2.5% can occur. However, this maximum error occurs right near the 50% point, where large errors are not normally important. If the player needs to figure out who is ahead when the race is close, the sign of D should be enough.

As an example, suppose $X = 70$ and $Y = 80$. We compute $\varDelta = 80 - (70 - 4) = 14$ and $S = 70 + 80 = 150$. Then, we compute the numerator, $\varDelta^2 + \varDelta/7 = 14^2 + 14/7 = 196 + 2 = 198$ and the denominator, $S - 25 = 125$.

Table 1. The table to use.

Π	Value of $\frac{\Delta^2+\Delta/7}{S-25}$	Π	Value of $\frac{\Delta^2+\Delta/7}{S-25}$
0.50	0.00000000	0.75	0.998131
0.51	0.00137882	0.76	1.09451
0.52	0.00551875	0.77	1.19769
0.53	0.0124302	0.78	1.30824
0.54	0.0221307	0.79	1.42679
0.55	0.0346450	0.80	1.55407
0.56	0.0500050	0.81	1.69092
0.57	0.0682506	0.82	1.83834
0.58	0.0894296	0.83	1.99749
0.59	0.113598	0.84	2.16975
0.60	0.140821	0.85	2.35678
0.61	0.171174	0.86	2.56060
0.62	0.204741	0.87	2.78365
0.63	0.241618	0.88	3.02902
0.64	0.281913	0.89	3.30059
0.65	0.325747	0.90	3.60337
0.66	0.373256	0.91	3.94399
0.67	0.424591	0.92	4.33145
0.68	0.479920	0.93	4.77844
0.69	0.539433	0.94	5.30360
0.70	0.603341	0.95	5.93596
0.71	0.671879	0.96	6.72439
0.72	0.745311	0.97	7.76102
0.73	0.823934	0.98	9.25404
0.74	0.908082	0.99	11.8737

Dividing, we get $198/125 = 1.584$; looking this up in the table, we find we are between 0.80 and 0.81, but closer to 0.80.

A less tidy example starts with $X = 70$ and $Y = 90$, so $\Delta = 24$ and $S = 160$. The numerator is then $24^2 + 24/7 = 576 + 3.43 = 579.43$, and the denominator is $S - 25 = 135$. Dividing, we get 4.292, and looking this up in the table gives a value between 0.91 and 0.92, but closer to 0.92. If we go back and round the numerator to 579, our resulting division gives $4.28\overline{8}$; even if we rounded up to 580, our division gives 4.296. In either case, our ultimate decision is not affected.

Our last example has the player on roll behind: $X = 90$ and $Y = 70$. We get $\Delta = -16$ and $S = 160$. The numerator is $256 + (-16)/7 = 253.714$, and the denominator is 135 as before. Our ratio is 1.879, which the table says is a WP between 0.82 and 0.83. However, we know that the player on roll is behind, so these WP values apply to the other player instead; the player on roll has a WP value between $1 - 0.82$ and $1 - 0.83$.

It is handy to have simple rules to test if the WP is above a certain percentage. Here are some approximations for use over the board. They have been taken from Figure 12.

- The WP is above 90% if $Y \geq 1.1X + 12$.
- The WP is above 80% if $Y \geq 1.1X + 4$.

These are not strictly accurate even for the data presented here, but they are approximately correct.

4 Conclusion and acknowledgment

We have examined an existing system for estimating winning probabilities in the single checker model, and used more advanced renewal theory to make it more accurate. We have also compared the SCM to simulated backgammon races and adjusted our model to produce an even more accurate method of estimating winning probabilities, without much increase in the mental difficulty of the method.

We are grateful to Bob Leary at the San Diego Supercomputer Center for his supervision of Andrew Ross during a 1995 Research Experience for Undergraduates (REU) project that produced much of our data.

References

1. Benjamin, Arthur and Ross, Andrew M. (1997) Enumerating backgammon positions: the perfect hash. *Interface: Undergraduate Research at Harvey Mudd College* **16** (1) 3–10.
2. Buro, Michael (1999) Efficient approximation of backgammon race equities. *International Computer Chess Association Journal* **22** (3) 133–142.
3. Cormen, Thomas H., Leiserson, Charles E., Rivest, Ronald L., and Stein, Clifford (2001) *Introduction to Algorithms*, 2nd Ed. MIT Press and McGraw-Hill.
4. Feller, William (1968) *An Introduction to Probability Theory and Its Applications, Vol. 1*. Wiley, New York.
5. Keeler, Emmett B. and Spencer, Joel (1975) Optimal doubling in backgammon. *Operations Research* **23** (6) 1063–1071.
6. Kleinman, Danny (1980) *Vision Laughs at Counting*. Los Angeles.
7. Orth, P. J. (1976) A comment on "Optimal Doubling in Backgammon." *Operations Research* **24** (6) 1179.
8. Tesauro, Gerald (2002) Programming backgammon using self-teaching neural nets. *Artificial Intelligence* **134** (1–2) 181–199.
9. Thorp, Edward O. (1975) Backgammon: part I, the optimal strategy for the pure running game. Presentation at the Second Annual Conference on Gambling, South Lake Tahoe, NV.

10. Thorp, Edward O. (2007) Backgammon: the optimal strategy for the pure running game. In: Ethier, Stewart N. and Eadington, William R. (eds.) *Optimal Play: Mathematical Studies of Games and Gambling*. Institute for the Study of Gambling and Commercial Gaming, University of Nevada, Reno, 237–265.

11. Tijms, Henk C. (1994) *Stochastic Models: An Algorithmic Approach*. Wiley, New York.

12. Zadeh, Norman (1977) On doubling in tournament backgammon. *Management Science* **23** (9) 986–993.

13. Zadeh, Norman and Kobliska, Gary (1977) On optimal doubling in backgammon. *Management Science* **23** (8) 853–858.