A new algorithm for the approximation of ICM equities in tournament poker

Ben Roberts

October 14, 2011

1 Introduction

The importance of ICM calculations is well known amongst tournament poker players. Apart from the obvious interest in being able to approximate one's equity at any given time, the implications of ICM calculations can affect players' optimal strategies. Furthermore, knowledge of ICM equities are particularly useful for final table deal-making in large MTTs.

Unfortunately, the theoretical computation of true ICM equities is generally understood to be an intractable problem. Instead, the current practice is to approximate the equities. However, the standard Malmuth-Harville (MH) algorithm for performing such approximations is known to be somewhat inaccurate, tending to overestimate the equity of short stacks and underestimate that of large stacks.

I present a new algorithm to approximate ICM equities, which through testing has consistently proved to be more accurate than the MH algorithm, as well as being computationally inexpensive. Furthermore, my new algorithm can be adapted to perform a quick approximation of equities in large player pools, which the MH algorithm cannot do in reasonable time.

2 A short comparison

I briefly outline a comparison between my new algorithm and the MH algorithm to illustrate the improvement in accuracy. I've taken an example from the website of ICM Cruncher[1], in which they display each of five players'

[1]http://www.pokercruncher.com/ipICMCruncher.html

equities in a SnG paying \$500, \$300, \$200 for 1st, 2nd and 3rd. In Table 1 the approximated equities are displayed along with the ‘true’ equities.[2]

Player	Chip stack	Equity (\$)		
		Malmuth-Harville	My algorithm	True equity
1	2000	115.6	108.4	108.1
2	4000	206.9	207.8	206.8
3	3000	164.7	160.9	159.9
4	6000	271.0	277.0	279.0
5	5000	241.8	246.0	246.3

Table 1: Comparison of ICM equity approximations

It is evident that for this example, my new algorithm calculates ICM equities with signifantly higher accuracy than the MH algorithm. One can observe the tendency of the MH algorithm to overestimate the equity of short stacks and underestimate that of large stacks, while my algorithm does not have the same weakness.

My algorithm can also be adapted to accommodate a larger player pool. As an example, suppose there are 40 players remaining in a satellite event, in which the first 20 finishers each win a \$100 ticket. I assigned them varying chip stacks in the range between the shortest stack of 400 to the chip leader with nearly 30,000, and display their equities in Figure 1 as calculated by my algorithm. Note that Figure 1 supports the intuition that the medium stacks ≈10,000 should be the most risk averse, while the short stacks and large stacks should treat chips almost as they would in a cash game.

3 Notation and Theory

Suppose there are n players. We denote each player i’s stack size by x_i, and the probability that he finishes in place s by $p_i(s)$. We know the probability that a player wins the tournament is proportional to his stack size as the game is assumed to be fair:

$$p_i(1) = \frac{x_i}{\sum_{j=1}^{n} x_j} . \tag{1}$$

[2]The ‘true’ equities are accurate estimates achieved through Monte Carlo simulation with a very large sample size. More details are given in Section 4.

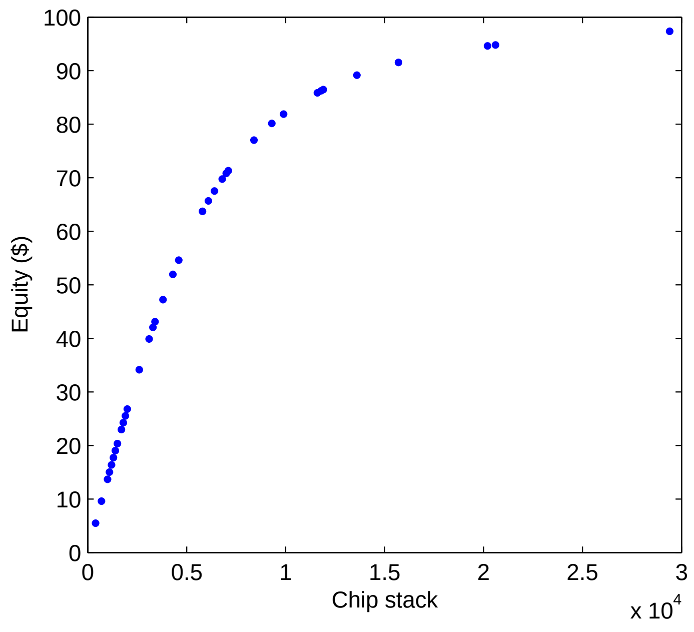

Figure 1: ICM equities in a large player pool

It is widely accepted that the theoretically correct values of the $\{p_i(s)\}$ are the appropriate absorption probabilities of a continuous random walk inside a $(n-1)$-dimensional tetrahedron with absorbing boundaries. This random walk is equivalent to repeatedly moving an infinitessimal amount of chips from one random player to another, whilst removing busted players from the game.

3.1 Malmuth-Harville algorithm

The MH algorithm works by setting the correct winning probabilities as in (1). It then operates under the assumption that if the top m places are taken up by a subset $\mathcal{S}$ of players, then the probability that player $i \notin \mathcal{S}$ finishes in $(m+1)$th is simply the probability that he would've won an exclusive tournament amongst the remaining players. Thus, the probability player i finishes 2nd is calculated to be

$$p_i(2) = \sum_{j \neq i} \left[p_j(1) \cdot \frac{x_i}{\sum_{k \neq j} x_k} \right].$$

The rest of the $p_i(s)$'s can be found in an iterative fashion. Note that this method can be quite computationally expensive as the calculation of $p_i(s)$ is the weighted average of $(n-1)!/(n-s)!$ terms.

3.2 My algorithm

My algorithm works in a similar fashion to a different algorithm — the Malmuth-Weitzman (MW) algorithm. The MW algorithm assumes that the probability that player i is next eliminated is inversely proportional to his chip stack, and if realized reallocates his chips evenly amongst the remaining players.

My algorithm makes a slightly different assumption about the probabilities of next elimination:

$$p_i(n) \propto \frac{1}{x_i^2} \cdot \sum_{j \neq i} \frac{1}{x_j} \,. \tag{2}$$

If eliminated, player i's chips are then distributed amongst the remaining players inversely proportionally to their stacks:

$$\mathbb{P}(j \text{ wins} \mid i \text{ next elim}) = \frac{x_j + \delta_{ij} x_i}{X} \,,$$

where

$$\delta_{ij} = \frac{x_j^{-1}}{\sum_{k \neq i} x_k^{-1}} \,,$$

and X is the total number of chips in play. Given that we redistribute a busted player's chips in this fashion, the elimination probabilities (2) are necessary to ensure that a player's chip stack is a martingale, which in turn ensures the correct winning probabilities (1).

An intuition of why a shorter stacked player should gain more of player i's chips given that i is next eliminated can be gained by realizing that the information is more valuable to him. For example, suppose there are three players left with chip stacks in the ratio of 1 : 1 : 10. We denote the probability that player 3 wins **given that player 1 is next eliminated** by $\tilde{p}_3(1)$. By symmetry,

$$\tilde{p}_3(1) = \frac{p_3(1)}{1 - p_3(3)} \,.$$

The probability that player 3 is next eliminated $p_3(3)$ is clearly very small, and thus $\tilde{p}_3(1)$ is very close to $p_3(1)$. This is reflected by transferring only a small fraction of player 1's chips to player 3 upon his elimination, while the bulk are transferred to player 2.

3.3 Example

Tom Ferguson (Chris' father) once wrote an unpublished paper describing how to analytically calculate the true values of the $\{p_i(s)\}$ for $n = 3$. His method is quite convoluted and doesn't generalize to $n \geqslant 4$. However, he did include the correct solution for the simplest non-trivial case where $\boldsymbol{x} = (1, 1, 2)$ — defined by the probability that the chip leader is next eliminated: $p_3(3) = 0.1421$.[3]

Malmuth-Harville. We start with the winning probabilities:

$$p_i(1) = \left(\tfrac{1}{4}, \tfrac{1}{4}, \tfrac{1}{2}\right).$$

If player 3 does not win, the probability he gets 2nd is assumed to be the probability he would've beaten one of the shorter stacks HU — 2/3. As he doesn't win the tournament half the time, we have $p_3(2) = 1/3$, and so by symmetry:

$$p_i(2) = \left(\tfrac{1}{3}, \tfrac{1}{3}, \tfrac{1}{3}\right).$$

This leaves us with the probabilities of next elimination to be:

$$p_i(3) = \left(\tfrac{5}{12}, \tfrac{5}{12}, \tfrac{1}{6}\right).$$

My algorithm. We calculate the probabilities of next elimination as per equation (2): $p_i(3) \propto (\tfrac{3}{2}, \tfrac{3}{2}, \tfrac{1}{2})$, giving

$$p_i(3) = \left(\tfrac{3}{7}, \tfrac{3}{7}, \tfrac{1}{7}\right).$$

If player 1 is eliminated first, his chips are allocated to the other two players in the ratio 2 : 1. That is, the new stacks would be $(\tilde{x}_2, \tilde{x}_3) = \left(\tfrac{5}{3}, \tfrac{7}{3}\right)$. By symmetry, the probability that player 3 finishes 2nd is therefore

$$\begin{aligned} p_3(2) &= \mathbb{P}(\text{he does not finish 3rd}) \times \mathbb{P}(\text{he loses the resulting HU battle}) \\ &= \frac{6}{7} \cdot \frac{(5/3)}{4} \quad = \frac{5}{14}. \end{aligned}$$

This gives us

$$p_i(2) = \left(\tfrac{9}{28}, \tfrac{9}{28}, \tfrac{5}{14}\right),$$

and leaves us with the correct winning probabilities

$$p_i(1) = \left(\tfrac{1}{4}, \tfrac{1}{4}, \tfrac{1}{2}\right).$$

[3] Tom's paper can be found at `www.math.ucla.edu/~tom/papers/unpublished/gamblersruin.pdf`

From this simple example, one can already observe the improved accuracy of my algorithm, as the resulting value for $p_3(3) = (1/7) = 0.1429$ is much closer to the true value (0.1421) than the MH algorithm which finds $p_3(3) = (1/6) = 0.1667$.

3.4 Adaptation to reduce computation time

In the form described in Section 3.2, my algorithm is of comparable computational complexity to the MH algorithm, as they both find the probabilities of every permutation of placings. However, my algorithm can be adapted in a way that is difficult to do with the MH algorithm.

This is achieved by iterating the following procedure for each player, although we describe it in reference to player 1.

1. Start with finding the probabilties of next elimination $p_i(n)$ as per equation (2).

2. Find the value $\tilde{x}_1$: defined as the expected stack of player 1 **given player 1 is not the next elimination**. That is,

$$\tilde{x}_1 = \frac{1}{1 - p_1(n)} \sum_{i \neq 1} \Big[p_i(n)(x_1 + \delta_{i1} x_i) \Big] .$$

3. Similarly, find the values of $\tilde{x}^{(m)}$ for $m = 1, \ldots, n-2$: defined as the expected size of the mth largest other stack after the first elimination **given player 1 is not the next elimination**. For example,

$$\tilde{x}^{(1)} = \frac{1}{1 - p_1(n)} \sum_{i \neq 1} \Big[p_i(n) \max_{j \neq 1, i} \{ x_j + \delta_{ij} x_i \} \Big] .$$

4. Calculate the new probabilities of next elimination using (2) assuming the new stacks are $\big(\tilde{x}_1, \tilde{x}^{(1)}, \ldots, \tilde{x}^{(n-2)}\big)$, and let's denote these probabilities by $\big(q_1, q^{(1)}, \ldots, q^{(n-2)}\big)$. The probability player 1 finishes in $(n-1)$th is then:

$$p_1(n-1) = q_1 \cdot (1 - p_1(n)) .$$

5. Go back to Step 2 assuming the adjusted stacks and elimination probabilities.

Adapting the algorithm in this fashion removes the need to calculate the probabilities of each permutation of player placings, reducing the complexity of the program to $O(n^3)$. Empirical testing has showed only very small differences between the results of this adaptation and the longer version.

4 A long comparison

4.1 Discretization

In order to compare the accuracy of my algorithm to the Malmuth-Harville algorithm, we need to identify certain cases in which we can be reasonably confident of the theoretically correct values that we are trying to approximate. One way to do this is to assume a discretization of the problem for cases in which the starting chip stacks have a large common factor.

In the discretized version of the problem, we assume random exchanges of a single chip between players instead of infinitessimal exchanges. For small enough cases it is easy to find the theoretical solution to the discretized problem by iterating a diffusion process over the state space describing the evolution of the probability distribution of the chip stacks over time. One nice thing we find is that the solution converges very quickly as we reduce the coarseness of the discretization.

For example, we consider the simplest non-trivial case studied by Tom Ferguson in which the three players have chip stacks proportional to $(1, 1, 2)$. Assuming random single chip exchanges with starting stacks $\boldsymbol{x} = (1, 1, 2)$, we find $p_3(3) = (1/7) = 0.14286$. With starting stacks $\boldsymbol{x} = (2, 2, 4)$ we find $p_3(3) = 0.14222$, and for $\boldsymbol{x} = (3, 3, 6)$ we find $p_3(3) = 0.14217$. It is clear that the solution is quickly converging to the true value of 0.1421. Other examples have also showed remarkable speed of convergence. This is nice because we can take a reasonably coarse discretization of a problem and expect the resulting solution to be very close to that of the continuous version.

4.2 Comparing probabilities

I ran simulations for numerous examples to compare the accuracy of my algorithm to the MH algorithm and without exception found mine to be significantly more accurate. Tables 2, 3, and 4 detail the results for three examples. The probabilties calculated from my algorithm are shown in blue, those calculated from the MH alorithm are shown in red, while the 'true' equities are shown in black.[4]

[4]We estimate the 'true' equities by assuming reasonable discretizations of the examples. For the first two example in which $n = 3$, we iterate a diffusion process as described in Section 4.1. For the first we assume starting stacks of $\boldsymbol{x} = (5, 5, 10)$, and for the second $\boldsymbol{x} = (2, 18, 20)$. For the third example, the state space is too large to perform a similar diffusion process. Instead we use crude Monte-Carlo estimation with 1 million samples, ensuring that the standard deviation of the estimates of each $p_i(s)$ is less than 0.05%.

Player	1st %	2nd %	3rd %
1	25	32.11	42.89
	25	32.14	42.86
	25	33.33	41.67
2	25	32.11	42.89
	25	32.14	42.86
	25	33.33	41.67
3	50	35.78	14.22
	50	35.71	14.29
	50	33.33	16.67

Table 2: $n = 3$, $\boldsymbol{x} \propto (1, 1, 2)$

Player	1st %	2nd %	3rd %
1	5	6.60	88.40
	5	5.47	89.53
	5	9.09	85.91
2	45	48.59	6.41
	45	49.24	5.76
	45	47.37	7.63
3	50	44.81	5.19
	50	45.29	4.71
	50	43.54	6.46

Table 3: $n = 3$, $\boldsymbol{x} \propto (1, 9, 10)$

Player	1st %	2nd %	3rd %	4th %	5th %
1	10	10.74	12.92	19.32	47.02
	10	10.59	13.32	20.61	45.58
	10	11.88	14.99	21.28	41.85
2	15	15.94	18.51	25.21	25.33
	15	15.79	19.24	26.20	23.76
	15	16.85	19.58	24.00	24.58
3	20	20.68	22.38	22.64	14.31
	20	20.69	22.87	22.08	14.36
	20	20.99	21.94	21.40	15.67
4	25	24.78	23.46	18.41	8.34
	25	24.93	23.08	17.41	9.57
	25	24.15	22.17	18.14	10.54
5	30	27.86	22.73	14.42	5.00
	30	28.00	21.48	13.69	6.83
	30	26.13	21.33	15.17	7.37

Table 4: $n = 5$, $\boldsymbol{x} \propto (2, 3, 4, 5, 6)$

By inspection, we can see that my algorithm generates approximate probabilities generally closer to the true values for all three examples. To quantify this, I calculated the average absolute error ε for each algorithm and example. That is,

$$\varepsilon = \mathbb{E}\left[\, |\widehat{p}_i(s) - p_i(s)| \,\right],$$

assuming a uniform distribution over the choices of (i, s). In the first example, we find that my algorithm produces near-perfect results, with average error $\varepsilon = 0.03\%$. In comparison, the MH algorithm produces an average error of $\varepsilon = 1.09\%$.

The difference is not as great in second and third examples but it is still significant. The average errors produced by my algorithm are 0.50% and 0.59% respectively, while the MH algorithm produces 1.11% and 1.13%. From looking at various examples like these, I found that my algorithm generally seems to reduce the average error by a factor of about 2 to 3.

4.3 Comparing equity

To illustrate how the improved accuracy manifests itself in equity calculation, we assign prize structures to the three previous examples. For the first two in which $n = 3$ we assume prizes for 1st and 2nd to be \$200 and \$100 respectively. For the third example we assume a \$500, \$300, \$200 prize structure as in the example described in Section 2.

Player	Equity (\$)		
	MH	Mine	True
1	83.3	82.1	82.1
2	83.3	82.1	82.1
3	133.3	135.7	135.8

Table 5: $\boldsymbol{x} \propto (1, 1, 2)$

Player	Equity (\$)		
	MH	Mine	True
1	19.1	15.5	16.6
2	137.4	139.2	138.6
3	143.5	145.3	144.8

Table 6: $\boldsymbol{x} \propto (1, 9, 10)$

Player	Equity (\$)		
	MH	Mine	True
1	115.6	108.4	108.1
2	164.7	160.9	159.9
3	206.9	207.8	206.8
4	241.8	246.0	246.3
5	271.0	277.0	279.0

Table 7: $\boldsymbol{x} \propto (2, 3, 4, 5, 6)$

Again, it is evident that my algorithm is significantly more accurante than the MH algorithm. In a similar fashion to Section 4.2, I quantify this by calculating the average absolute error

$$\varepsilon = \frac{1}{n}\sum_{i=1}^{n} |\widehat{y}_i - y_i| \,,$$

where y_i is player i's true equity. We find that the average errors of my algorithm for the three examples are respectively 0.05, 0.75, and 0.95, while the MH alorithm produces average errors 1.63, 1.66, and 4.99. By investigating other examples, I've found that my algorithm generally reduces the absolute error of the equity approximations by a factor of about 3 to 5.