Efficient Approximation of Backgammon Race Equities

Michael Buro

NEC Research Institute
4 Independence Way
Princeton NJ 08540, USA

Abstract

This article presents efficient equity approximations for backgammon races based on statistical analyses. In conjunction with a 1–ply search the constructed evaluation functions allow a program to play short races almost perfectly with regard to checker–play as well as doubling cube handling. Moreover, the evaluation can naturally be extended to long races without losing much accuracy.

1 Introduction

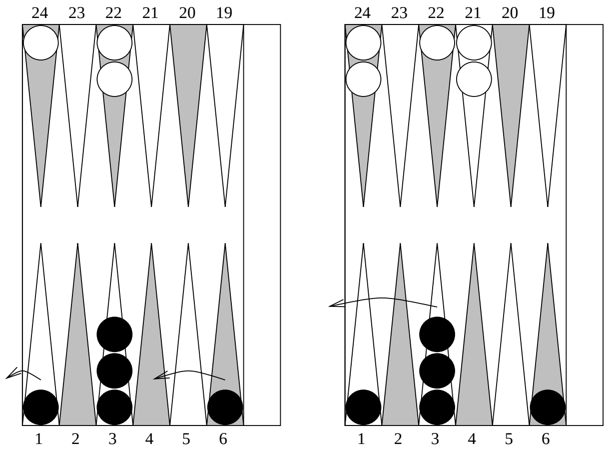
Several popular games can end up in race positions in which both players reach their goals separated from each other. Since the search and evaluation complexity in these positions is much lower than for positions in which the playing parties can influence each other, finding optimal moves may be feasible even under tournament timing constraints. We call (end-)game positions *separable* if future optimal moves of a player do not depend upon the opponent's moves. Below we list some examples of games in which separable positions occur:

- Domineering and Amazons. In these games, which are described in [4], both players try to create territory to which they have sole access. Once a position has been reached without disputed territory the players make as many moves as possible in their own territory in order to avoid running out of moves before the opponent. Determining the winner in these situations is simple as it only requires a static board analysis.
- Halma and Chinese Checkers. Playing on a star–shaped grid, up to six players try to move all their marbles to the opposite side of the board as fast as possible. Once a marble group is disengaged from the other groups finding optimal moves turns into a simpler single–agent search problem.
- Simplified backgammon. Before this game begins, so called *pip–counts*[1] are assigned to both players who then alternately roll two dice and decrease their pip–counts by the total number of points rolled. As in backgammon doubles count twice. The winner is the first player who reaches a pip–count less than one. Obviously, in this game all positions are separable because in each position only one move is possible which solely depends on the player's rolls so far.

The alert reader may ask why backgammon[2] is not on this list. This game also turns into a race when the checker groups are separated. Each player then tries to bear–off all his checkers as quickly as possible – seemingly independent of the opponent's checker configuration. In general, however, it turns out that backgammon positions are not separable even if the checker groups are separated. There are several reasons for this phe-

[1] A backgammon term which denotes the total number of points (or pips) that a player must move to bear–off all checkers.

[2] Backgammon rules and glossaries can be found on the WWW. A good introduction of basic game strategies is given in [3].



Black to play (1,2)

	move	P(win)		move	P(win)
a)	6-4 1-off	7.48%	b)	3-off	33.37%
	3-off	2.58%		6-4 1-off	29.82%

Figure 1: a) A race position with presumably the greatest min–ENR move error ($\Delta P(\text{win}) = 4.90\%$). The min–ENR move 3-off is a blunder which reduces Black's winning chance by a factor of almost three. b) A similar position where 3-off is better ($\Delta P(\text{win}) = 3.55\%$).

nomenon:

1. The player who first bears–off all checkers wins at backgammon. Intuitively, one would therefore expect that minimizing the expected number of rolls (ENR) maximizes the expected payoff (equity). There are, however, positions where this intuition fails (Fig. 1a). Black's only (desperate) winning plan is to hope for rolling high doubles next and to prepare for it by playing 6-4 1-off. Although this move does not minimize the ENR, it almost triples Black's winning chance. By contrast, in a similar position presented in Fig. 1b) the min–ENR move 3-off is best. Apparently, the chance component in backgammon causes playing interactions in race positions even though the checker groups are separated.

2. If a player has gained a considerable race advantage he may be able to bear–off all of his checkers before the opponent has got out his first one. This ending is called a *gammon* and counts two points. Depending on the opponent's position one therefore has to decide whether to go for a win or to minimize the chance of being gammoned. Thus, in these situations optimal play is not independent of the opponent's checker configuration either.

3. In backgammon the stake of the current game can be doubled several times using a (doubling) cube with numbers 2, 4, 8, 16, 32, and 64 on its faces. A game starts with the cube placed in the middle of the board indicating that both players have the right to double and the game value currently is one. If at one point in the game a player having the right to double thinks he has a considerable advantage, he may double before his roll. The opponent can choose to decline the double, in which case the game ends with the current cube score. If he chooses to take the double, the cube is turned over to him showing the next power of two. The sole right to double next is now with the opponent and the game resumes with the player to move throwing the dice. Because proper cube handling depends on accurate winning chance estimation, again the opponent's checkers cannot be ignored.

Despite the fact that backgammon race positions in general are not separable, it is worthwhile to find out if there are both accurate and efficient approximation algorithms that perform sufficiently well in practice. The following two main sections deal with the fast approximation of short race equities without and in presence of the doubling cube. A discussion of applications and ideas on extensions with regard to long races and gammons concludes the paper.

2 Cubeless Equity

In simplified backgammon the winning chance for the side to move can be computed exactly provided that the distribution of the roll number needed for reaching a pip–count less than one is known. Let $X(c)$ be the random variable which measures the number of rolls to reach a pip–count < 1 when starting with pip–count c. Then, in a position defined by two pip–counts (c_1, c_2), player 1 to move wins with probability $P(\text{win}) = P(X(c_1) \leq X(c_2))$. This is equivalent to

$$P(\text{win}) = \sum_i x_i(c_2) \cdot \sum_{j \leq i} x_j(c_1),$$

where $x_i(c) := P(X(c) = i)$. Starting with pip–count 0, $x_i(c)$ can be computed using the following recursive relation:

$$\begin{aligned} x_0(c) &:= (c = 0) \\ x_{i+1}(c) &= \sum_{r=1}^{21} p_r \cdot x_i(\max(0, c - c_r)) \end{aligned}$$

There are 21 distinct rolls in backgammon. p_r denotes the probability and c_r the number of points of roll r.

Extending this roll distribution computation to real backgammon races, where one can choose among several move alternatives, is not hard. Let d be the checker configuration for the player to move and $x(d)$ the density function of $X(d)$ which now measures the number of rolls needed to bear–off all checkers when starting with configuration d and following the min–ENR strategy. Stated in a compact form it follows:

$$x(d) = S\big(\sum_{r=1}^{21} p_r \cdot x(d_r)\big),$$

where d_r is a successor configuration of d that leads to the lowest ENR for roll r, and operator S shifts a density function one place to the right. Beginning with the empty configuration ϵ, for which we define $x(\epsilon)(i) := (i = 0)$, all $x(d)$ may be computed efficiently again by dynamic–programming.

In actual game play we utilize this precomputed information in two ways. First, a min–ENR move vector can be assigned to each checker configuration. This allows us to play plausible moves instantly without considering the opponent's checker configuration. Although min–ENR moves are not always perfect and checker play may depend on the opponent's configuration, this strategy may still work well on average. Second, in a position given by two checker configurations (d_1, d_2) the roll distribution information enables us to approximate the winning chance of player 1 to move, namely

$$\hat{P}(\text{win}) = P(X(d_1) \leq X(d_2)). \quad (1)$$

Table 1: Number of configurations of up to n checkers of one color distributed on up to m points.

#checkers $n \leq$	#points $m \leq$	#configurations
6	6	924
7	6	1,716
8	6	3,003
9	6	5,005
10	6	8,008
11	6	12,376
15	6	54,264
15	11	7,726,160
15	12	17,383,860
15	23	1,708,582,624
7	23	2,035,800
8	23	7,888,725

This approximation may be used in a look–ahead search to determine good moves.

In order to gauge the performance of both approaches — which seem to be common knowledge among backgammon programmers, but never found their way into scientific literature in the past — we gathered statistics and played a series of tournaments against optimal players. Looking at the configuration counts given in Table 1 the construction of optimal players by computing a two–dimensional array of winning probabilities is infeasible even for moderate configuration sizes. For instance, covering all race positions with up to 15 checkers distributed on up to 6 points for either side requires the calculation of $54264^2 \approx 3 \cdot 10^9$ probabilities. Therefore, we decided to restrict experiments to relatively small configuration sizes (up to 10 checkers on up to 6 points) and to use a series of (tournament) statistics demonstrating how the performance changes when the pip–counts are increased.

The optimal players for various configuration sizes were constructed by encoding the players' checker configurations into a pair of integers and using these indices to access a square array of winning probabilities. The array was then filled by a recursive procedure similar to the one used for computing roll distributions. Given a position and a roll the optimal player just picks the move which minimizes the opponent's winning probability.

The statistics presented in Fig. 2 indicate that $\hat{P}(\text{win})$ is a very good approximation to $P(\text{win})$. Large errors of up to 1.39% are rare. On average, $\hat{P}(\text{win})$ only slightly underestimates the winning chance in bad positions and slightly overestimates it in good positions. A plausible explanation for this behavior is that sometimes the (probably) losing player has desperate but effective move options similar to that presented in Fig. 1a). Since the standard deviation of the error is also small, a good performance of a 1–ply search player, which uses $\hat{P}(\text{win})$ as evaluation function, can be expected. An interesting ob-

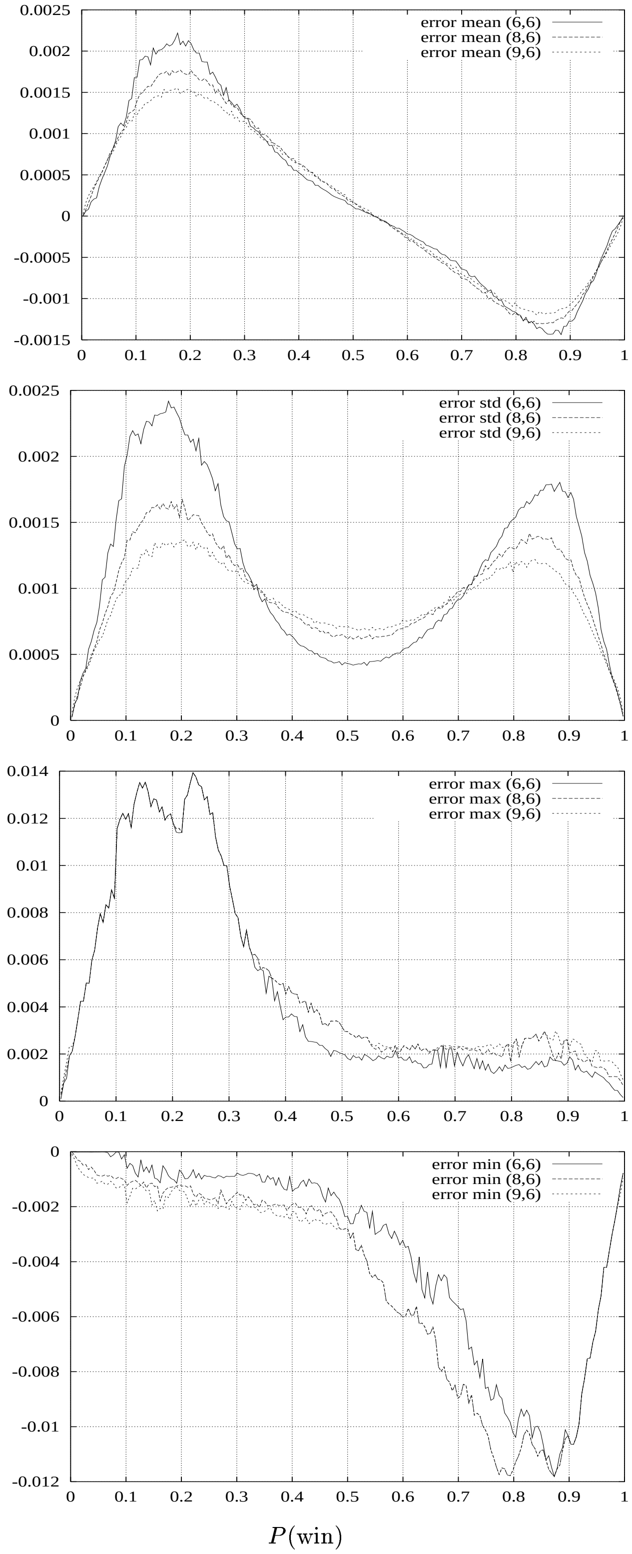


Figure 2: Statistics of the evaluation error $e = P(\text{win}) - \hat{P}(\text{win})$ for several position sets (up to 6, 8, or 9 checkers on up to 6 points for either side). The data has been grouped using 200 intervals. The graphs show the mean, standard deviation, maximum, and minimum of e.

servation is that both the maximum error mean and the maximum standard deviation decrease when considering positions with higher pip–counts. Moreover, the maximum errors do not change much anymore when moving from (8,6) to (9,6) positions. This is a strong indication that $\hat{P}(\text{win})$ becomes more accurate for increasing pip–counts.

To check the actual game playing performance, we conducted experiments in form of tournaments. Starting with close random positions we let the min–ENR player and the 1-ply player play a large number of games against (nearly) optimal opposition. Table 2 presents the results. Two heuristic players (0-ply, 1-ply) played a series of tournaments against two nearly optimal players (opt-9, opt-16). 0-ply followed the min–ENR strategy, whereas 1-ply performed 1-ply searches utilizing the winning chance approximation (1). opt-k played moves leading to the highest winning probability. These probabilities were encoded using k bits *while* computing the "perfect-play" data. Each tournament consisted of 100,000 game pairs. The starting positions were chosen randomly according to the configuration parameters (#checkers $\leq n$, #points $\leq m$). Close starting positions and sufficient game lengths were enforced by only accepting positions with $47\% \leq P(\text{win}) \leq 53\%$ and pip–count $\geq \frac{3}{5} \cdot$ maximum pip–count for either side ($n \cdot m$). To further reduce the result's variance both games of a pair were played using identical roll sequences. The average game length is measured in half–moves, game payoffs were $+1$ and -1, and the average payoff standard deviations were estimated by a re–sampling procedure. As predicted by the excellent error statistics the players which are guided by heuristics perform very well. In particular, the min–ENR player ("0-ply") only loses around 0.002 points per game when starting with (10,6) configurations and playing against an optimal opponent. "1-ply" is even better: neither a statistically significant performance difference compared to the optimal player can be detected even when playing as much as 200,000 games, nor does "1-ply" seem to get weaker when the pip–count is increased. Hence, for practical purposes looking 1-ply ahead and applying winning chance approximation (1) at the leaves is sufficient for nearly optimal checker play in short backgammon races. Moreover, the error statistics and tournament results indicate that the 1-ply approach can be extended to longer races without losing performance. Table 2 also shows that encoding of the exact winning probabilities using nine bits is sufficient in practice. As we shall see in the next section, limiting the resolution is essential for a space efficient construction of cube equity arrays.

Table 2: Tournament results.

Config. size (chks.,pts.)	Player A	Player B	avg. game length	A's avg. payoff/game	std. dev.
(6,6)	0-ply	opt-9	6.28	−0.00105	0.00026
(8,6)	0-ply	opt-9	8.19	−0.00155	0.00030
(9,6)	0-ply	opt-9	9.21	−0.00151	0.00031
(10,6)	0-ply	opt-9	10.21	−0.00227	0.00031
(6,6)	0-ply	opt-16	6.28	−0.00100	0.00025
(8,6)	0-ply	opt-16	8.19	−0.00164	0.00029
(9,6)	0-ply	opt-16	9.21	−0.00162	0.00030
(6,6)	1-ply	opt-9	6.29	−0.00014	0.00013
(8,6)	1-ply	opt-9	8.20	+0.00017	0.00017
(9,6)	1-ply	opt-9	9.22	+0.00006	0.00017
(10,6)	1-ply	opt-9	10.22	−0.00015	0.00018
(6,6)	1-ply	opt-16	6.29	−0.00008	0.00008
(8,6)	1-ply	opt-16	8.20	+0.00011	0.00010
(9,6)	1-ply	opt-16	9.21	−0.00003	0.00010

3 Cube Equities

The doubling cube was introduced in backgammon in the 1920s. It adds a new skill to the game and its proper handling in k-point matches and money games is crucial. In presence of the doubling cube the expected game payoff depends not only on the checker configuration, but also on the current cube value and the right to double next. Moreover, cube actions are also influenced by the gammon potential and the current match standings. This section deals with cube equities in short race positions and their efficient approximation. As in the previous section, we will first develop a theoretical basis for the equity computation and then gauge the performance of simple approximation algorithms by means of error statistics and tournaments.

Keeler & Spencer [2] started their investigation of the

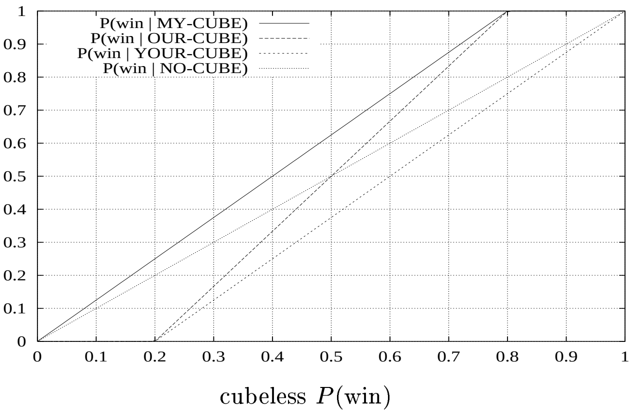


Figure 3: Cubeful winning probabilities depending on doubling rights and the cubeless winning probability in continuous games. MY-CUBE, OUR-CUBE, and YOUR-CUBE denote the cases in which the player to move has the sole right to double, both players have doubling right, or the opponent has the cube.

optimal doubling strategy in money games by considering continuous games, in which the winning probability is a continuous function of the (continuous) playing time. In those games, the doubling, redoubling, and take/drop decisions solely depend on the current winning probability of the player to move: he should double iff $P(\text{win}) \geq 0.8$ and the opponent should accept the double iff $P(\text{win}) \leq 0.8$. Fig. 3 visualizes the general relationship between the cubeless winning chance $P(\text{win})$ and winning probabilities when using a doubling cube in continuous games. Keeler & Spencer then went on transferring the optimal doubling strategy of simplified backgammon, which solely depends on pip–counts and can be computed by dynamic–programming, to backgammon. However, this simple doubling strategy performs poorly in short races, because in this stage backgammon games are far from being continuous and pip–counts alone are no reliable winning chance predictors.

One way to improve this approach is to predict the winning chance more accurately and to refine the cube equity model. In order to study the relationship between winning chances in continuous and discrete games we constructed optimal players for moderate configuration sizes and computed statistics which are presented in Fig. 4. Although there is a qualitative resemblance to the winning chance graph for continuous games, the top graph in Fig. 4 reveals remarkable nonlinear deformations. These are caused by possibly large winning chance variations in low pip–count positions. Even though the winning probability might be low before rolling the dice, a high roll can easily turn the table in short races. Hence, reaching a winning chance lower than 20% does not necessarily imply that the opponent wins the game in his next turn by doubling — as it is the case for continuous games. This explains the increased cube equities in the left part of the graph. Another interesting observation is that on average the doubling point is much less than 80%. Again, large winning chance variations are responsible. Prominent examples are last roll situations in which the player to move should double if his winning chance is better than 50%.

For increasing pip–counts backgammon races get more continuous in the sense that winning chance fluctuations get smaller. Consequently, the cube equity functions converge to those shown in Fig. 3. This observation allows us to approximate cube equities and proper cube handling efficiently. The most simple approach is to determine cube actions by utilizing estimated double/take functions in conjunction with the cubeless winning chance approximation (1). We call this kind of algorithm "C-0-ply." The C-0-ply player we implemented moves checkers according to the min–ENR strategy. For cube handling it utilizes winning chance approximation (1) in conjunction with estimated double/take functions $f_{\text{MY-CUBE}}, f_{\text{OUR-CUBE}}, f_{\text{TAKE}}$ (shown in the bottom graph of Fig. 4) as follows:

$$
\begin{array}{lll}
\text{my cube:} & \text{double} & \Leftrightarrow \quad r \leq f_{\text{MY-CUBE}}(\hat{P}(\text{win})) \\
 & \text{take} & \Leftrightarrow \quad r \leq f_{\text{TAKE}}(\hat{P}(\text{win})) \\
\text{our cube:} & \text{double} & \Leftrightarrow \quad r \leq f_{\text{OUR-CUBE}}(\hat{P}(\text{win})) \\
 & \text{take} & \Leftrightarrow \quad r \leq f_{\text{TAKE}}(\hat{P}(\text{win}))
\end{array}
$$

$r \in [0, 1]$ is a realization of a uniformly distributed random variable. This randomized algorithm slightly generalizes Keeler & Spencer's approach which uses pip–count thresholds for determining cube actions. For the exper-

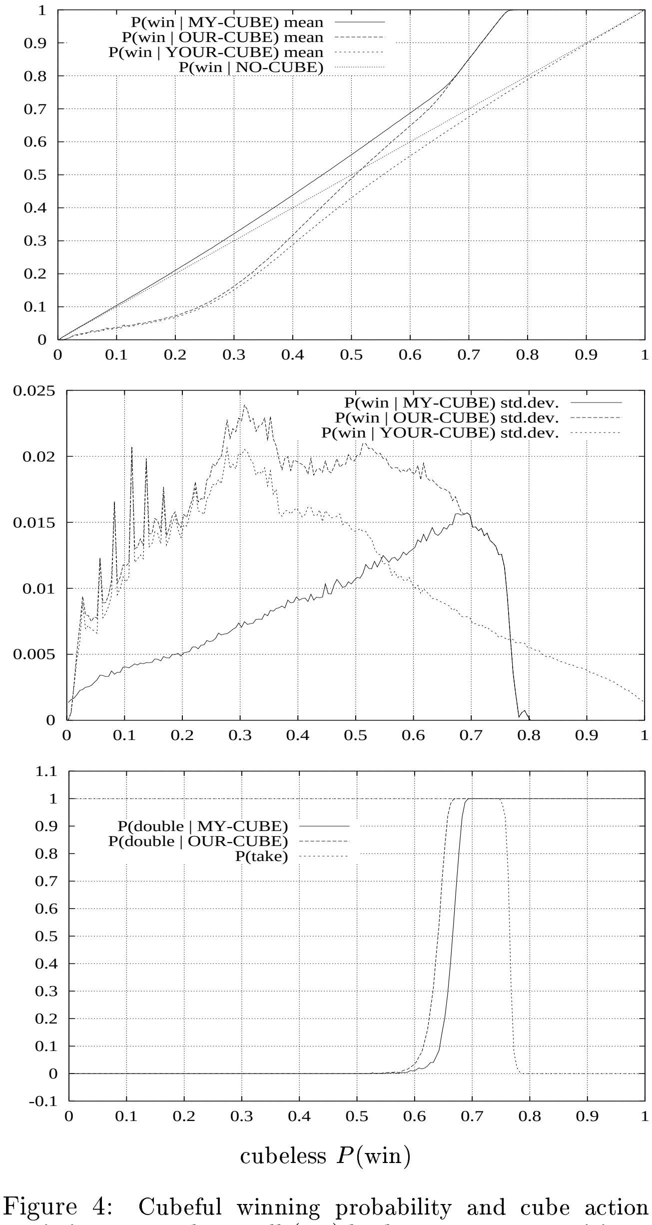


Figure 4: Cubeful winning probability and cube action statistics averaged over all (6,6) backgammon race positions. Again, the cubeless winning percentage was used to group the data into 200 intervals.

Table 3: Cube tournament results. Again, 200,000-game tournaments were played to compare the playing strength of several algorithms. Starting positions and roll sequences were picked as before. The initial cube value was one and both players had the right to double first.

Config. size (chks.,pts.)	Player A	Player B	avg. game length	A's avg. payoff/game	std. dev.
(6,6)	C-0-ply	C-opt-16	5.19	−0.0901	0.0008
(10,6)	C-0-ply	C-opt-9	8.09	−0.0729	0.0010
(6,6)	C-1-ply	C-opt-16	5.11	−0.0056	0.0006
(10,6)	C-1-ply	C-opt-9	7.92	−0.0070	0.0008
(10,6)	C-1-ply*	C-opt-9	7.93	−0.0006	0.0006
(6,6)	C-opt-9	C-opt-16	5.14	−0.00006	0.00011

iments reported in Table 3 we equipped C-0-ply with the estimated double/take functions for (6,6) and (10,6) positions and let it play tournaments against optimal opposition. The playing performance is not convincing: C-0-ply in average loses around 9 points in 100 (6,6) games and around 7.3 points in 100 (10,6) games.

Looking ahead may be crucial because of the non–linear relation between cubeless and cubeful winning chances. "C-k–ply" algorithms deal with this problem by assigning three equities (e_{my}, e_{our}, $e_{\text{your}} \in [-1, 1]$) to leaf nodes at depth $k > 0$ and propagating these values up the search tree using the following negamax cube recursion:

$$\begin{aligned} e_{\text{my}} &= \max\left\{ \min\{1,\ 2 \cdot e'_{\text{your}}\},\ e'_{\text{my}} \right\} \\ e_{\text{our}} &= \max\left\{ \min\{1,\ 2 \cdot e'_{\text{your}}\},\ e'_{\text{our}} \right\} \\ e_{\text{your}} &= e'_{\text{your}} \end{aligned}$$

e'_{my}, e'_{our}, and e'_{your} denote the average negated cube equities of the best successor positions depending on the right to double. The max/min operations correspond to the double/take choices of the two players. In interior doubling nodes the heuristically best cube actions are given by the following relations:

$$\begin{array}{lll} \text{my cube:} & \text{double} \Leftrightarrow & e_{\text{my}} \geq e'_{\text{my}} \\ & \text{take} \Leftrightarrow & \neg\text{double} \vee e'_{\text{your}} < 0.5 \\ \\ \text{our cube:} & \text{double} \Leftrightarrow & e_{\text{our}} \geq e'_{\text{our}} \\ & \text{take} \Leftrightarrow & \neg\text{double} \vee e'_{\text{your}} < 0.5 \end{array}$$

In order to improve upon the C-0-ply algorithm we constructed a C-1-ply player as follows. Checkers are played by looking one ply ahead and picking the move which minimizes the opponent's cubeless winning chance approximation (1). For determining cube actions, C-1-ply assigns three equities e_{my}, e_{our}, and e_{your} to chance nodes at depth one by mapping $\hat{P}(\text{win})$ to equities according to the estimated cube functions (Fig. 4 (top)).[3]

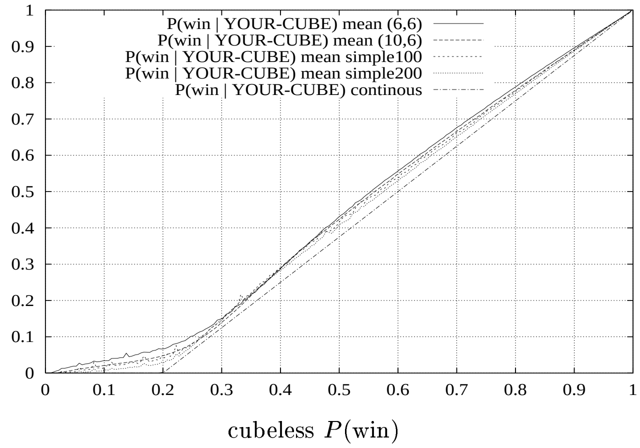


Figure 5: Average winning chance functions for (6,6) and (10,6) backgammon positions compared to simplified backgammon (pip–counts ≤ 100, 200 for either side) and continuous games. The opponent has the cube.

C-1-ply then propagates the values up to the root and picks the heuristically best action as described above. The tournament results summarized in Table 3 indicate that the average loss of C-1-ply is about 16 times smaller compared to C-0-ply in (6,6) tournaments and about 10 times smaller in (10,6) tournaments. This clear advantage shows the importance of looking ahead for proper cube handling. Moreover, only losing about 5.6 respectively 7 points on average in 1000 games comes close to optimal play and seems to be acceptable for practical purposes.

The optimal player that can handle (6,6) positions has access to a 924 × 924 array in which three cube equities and double/take flags are stored for each position. Assuming a 16–bit resolution the array thus occupies around 5.5 MB. While this array easily fits into the memory of today's PCs, increasing the configuration size quickly exhausts main memory or even hard discs. For instance, since there are 8,008 (10,6) configurations for either player, the according array occupies around 417 MB. By reducing the resolution to 9 bits when computing equities and applying simple compression techniques[4] the given space figures can be reduced by a factor of nearly 3. While this technique enables us to construct a nearly perfect (10,6) player and to hold the relevant data in memory for fast access, it is infeasible for larger configurations and we need to find accurate equity approximations.

As mentioned earlier simplified backgammon becomes a better backgammon model for increasing pip–counts. Moreover, we expect backgammon races to get "more

[3] In this case the relationship between equity e and winning probability P is $e = 2 \cdot P - 1$ since the only game outcomes are -1 and $+1$.

[4] Huffman encoding of equity differentials.

continuous" as well. Empirical evidence for this intuition is presented in Fig. 5. We constructed a simplified backgammon player, which handles the doubling cube perfectly, and compared the estimated winning chance functions to those for short backgammon races and continuous games. Apparently, simplified backgammon is also an excellent model for backgammon with regard to the doubling cube. This observation allows us to construct strong backgammon race players for longer races without relying on huge pre-computed arrays. The idea is to extend C-k-ply players by two components: a core that plays short races perfectly and a module that can handle longer races by adjusting leaf evaluations depending on pip–counts. One way to adjust the leaf evaluations e_{my}, e_{our}, and e_{your} is to interpolate values between two extreme cube functions: one for low pip–counts estimated from perfect play equities and one for high pip–counts obtained by averaging simplified backgammon equities.

Our experimental player C-1-ply* is equipped with a perfect (6,6) core. It evaluates non-(6,6) leaf positions p according to the following convex combination:

$$e_{\text{my}} = \alpha \cdot e_{\text{my}}^{(6,6)}(\hat{P}(\text{win})) + (1-\alpha) \cdot e_{\text{my}}^{\text{sim-200}}(\hat{P}(\text{win})),$$

where $e_{\text{my}}^{(6,6)}$ and $e_{\text{my}}^{\text{sim-200}}$ are the average cube equity functions for (6,6) backgammon and 200–pips simplified backgammon races in case the player to move owns the cube. The remaining cube equities e_{our} and e_{your} are computed analogously. $\alpha \in [0,1]$ is a weight which models the transition from low to high pip–count positions. It is piecewise linear in the total pip–count c of p :

$$\alpha = \begin{cases} 1, & \text{if } c \le 30 \\ (100-c)/(100-30), & \text{if } 30 < c < 100 \\ 0, & \text{if } c \ge 100 \end{cases}$$

The pip–count limits were picked by maximizing the tournament result against a fixed player. In order to gauge the playing strength of C-1-ply* we pitted it against the nearly optimal (10,6) player C-opt-9. The convincing tournament result is reported in Table 3: C-1-ply* is much better than C-1-ply. It almost reaches the performance of C-opt-9 although it has access to far less pre–computed data (4 MB rather than 146 MB (compressed)) and only uses a negligible amount of time for computing moves and cube actions (less than 0.1 seconds on a PII/333 PC).

These promising results for short races encourage to extend the approach to arbitrary race positions. Although the direct comparison with optimal players is no longer feasible due to space and time limitations, parameters can be optimized relative to fixed players, as mentioned above.

4 Discussion and Outlook

This article continues and extends earlier work on backgammon races. Keeler & Spencer [2] and Zadeh & Kobliska [6] studied optimal doubling strategies in the context of continuous games and simplified backgammon. They then developed cube action heuristics for backgammon races solely based on current pip–counts. With the advent of fast computers equipped with large memory it is possible today to construct perfect players for moderate configuration sizes. Our experiments in form of tournaments against optimal players indicate that the classical approach, which just examines the current position, is much weaker than 1-ply searches utilizing accurate winning chance predictions in conjunction with estimated cube functions. The equity approximation is in fact so accurate that the heuristic player comes close to perfection while using only a small fraction of the space needed for storing optimal cube equities.

A few years ago, Gerry Tesauro [5] developed an alternative doubling algorithm based on the position's *volatility* v, which is defined as the equities' standard deviation. Basically, his algorithms offers a double if the equity exceeds a specific doubling threshold $t(v)$. In a forthcoming publication he will describe his findings in detail which then enables us to compare both approaches.

Berliner's BKG-9.8 [1] was the first backgammon program which used pre–computed tables for guiding both checker–play and cube actions. Although the author was under the impression that min–ENR moves are optimal, his heuristic was not bad either as our tournament results show. Looking only one ply ahead in conjunction with a straight forward equity approximation, however, improves checker–play considerably. Our tournament results show that this simple well known heuristic indeed leads to nearly optimal play.

The main theme of this article has been the construction of equity approximations by utilizing statistics that become more accurate for increasing pip–counts. This has enabled us to create a nearly perfect hybrid player which makes use of pre–computed perfect information regarding "chaotic" low pip–count positions and is guided by accurate approximations in the high pip–count case. Extending this technique to cover all race positions in both k–point matches and money games is ongoing work. The main obstacle is the presence of gammons which complicates checker–play and cube handling. A possible solution to this problem may be the adjustment of the cube functions depending upon gammon potential and the current match standings.

References

[1] H.J. Berliner. BKG – a program that plays backgammon. In D.N.L. Levy, editor, *Computer Games I*, pages 3–28. Springer–Verlag, 1988.

[2] E.B. Keeler and J. Spencer. Optimal doubling in backgammon. *Operations Research*, 23(4):1063–1071, 1975.

[3] P. Magriel. *Backgammon*. 1976.

[4] R. Nowakowski, editor. *Games of No Chance*. Cambridge U Press, 1996.

[5] G. Tesauro. Personal communication. July 1999.

[6] N. Zadeh and G. Kobliska. On optimal doubling in backgammon. *Management Science*, 23(8):853–858, April 1977.