A Retrograde Approximation Algorithm for Two-Player Can't Stop

James Glenn[1], Haw-ren Fang[2], and Clyde P. Kruskal[3]

[1] Department of Computer Science
Loyola College in Maryland
4501 N Charles St., Baltimore, Maryland 21210, USA
jglenn@cs.loyola.edu
[2] Department of Computer Science and Engineering
University of Minnesota
200 Union St. S.E., Minneapolis, Minnesota, 55455, USA
hrfang@cs.umn.edu
[3] Department of Computer Science
University of Maryland
A.V. Williams Building, College Park, Maryland 20742, USA
kruskal@cs.umd.edu

Abstract. A two-player, finite, probabilistic game with perfect information can be presented as a four-partite graph. For Can't Stop, the graph is cyclic and the challenge is to determine the game-theoretical values of the positions in the cycles. In a previous paper we have presented our success on tackling one-player Can't Stop. In this paper we prove the existence and uniqueness of the solution to two-player Can't Stop, and present a retrograde approximation algorithm to solve it by incorporating the 2-dimensional Newton's method with retrograde analysis. We give results of small versions of two-player Can't Stop.

1 Introduction

Retrograde analysis has been well developed for deterministic and two-player zero-sum games with perfect information, and successfully applied to construct endgame databases of checkers [7, 9], chess [10, 11], and Chinese chess [1, 2, 13]. It also played a crucial role in solving Nine Men's Morris [3] and Kalah [6]. A recent success of parallel retrograde analysis was solving Awari [8].

On the other hand, retrograde analysis for probabilistic games is currently under-explored. Glenn [4] and Woodward [12] solved one-player Yahtzee. We presented our success on tackling one-player Can't Stop[1], by incorporating Newton's method into a retrograde algorithm run on a bipartite graph representing the game [5].

[1] Can't Stop was designed by Sid Sackson and marketed first by Parker Brothers and now by Face 2 Face Games. The rules can be found at http://en.wikipedia.org/wiki/Can't_Stop and http://www.boardgamegeek.com/game/41.

A two-player probabilistic game can be represented as a four-partite graph $G = (U, V, \bar{U}, \bar{V}, E)$, where $U/\bar{U}$ correspond to random events and $V/\bar{V}$ correspond to deterministic events of the first/second players, respectively. For Yahtzee, the graph representation is acyclic that simplifies algorithm design. In some games, such as Can't Stop, the graph representation is cyclic, which causes difficulty in designing a bottom-up retrograde algorithm. In this article we generalize our retrograde approximation algorithm for one-player Can't Stop by incorporating the 2-dimensional Newton's method into a retrograde algorithm.

The organization of this paper is as follows. Section 2 formulates the problem. Section 3 proves that two-player Can't Stop has a unique solution, and gives a retrograde algorithm to solve it. Section 4 presents the indexing scheme. Section 5 summarizes the results of the experimental tests. A conclusion is given in Section 6.

2 Problem Formulation

A two-player probabilistic game can be presented as a four-partite graph $G = (U, V, \bar{U}, \bar{V}, E)$, where $E \subseteq (U \times V) \cup (\bar{U} \times \bar{V}) \cup ((V \cup \bar{V}) \times (U \cup \bar{U}))$. Edges in $U \times V$ and $\bar{U} \times \bar{V}$ represent the random events (e.g., dice rolls), and edges in $V \times (U \cup \bar{U})$ and $\bar{V} \times (U \cup \bar{U})$ represent the deterministic events (i.e., the moves), by the first and second players, respectively. We also call vertices in $U \cup \bar{U}$ *roll positions* and vertices in $V \cup \bar{V}$ *move positions*. A terminal vertex indicates the end of a game. Without loss of generality, we assume all terminal vertices are roll positions (i.e., in $U \cup \bar{U}$).

A *partial turn* (u_1, u_2) (from roll position u_1 to roll position u_2) consists of a random event followed by a move. It is represented by a pair of edges $((u_1, v), (v, u_2))$ in G. A sequence of partial turns $(u_0, u_1), (u_1, u_2), \ldots, (u_{k-1}, u_k)$ is called a *turn*. In Can't Stop, a turn may consist of many partial turns. In Yahtzee a turn consists of exactly one partial turn, and hence $E \subseteq (U \times V) \cup (V \times \bar{U}) \cup (\bar{U} \times \bar{V}) \cup (\bar{V} \times U)$.

We associate each position with a real number representing the expected game score that the first player achieves in optimal play, denoted by a function $f : U \cup \bar{U} \cup V \cup \bar{V} \to R$. For two-player Can't Stop, $f(u)$ indicates the probability that the first player will win the game. Since the games we consider are zero-sum, the probability that the second player wins is $1 - f(u)$. For any terminal position $z \in U \cup \bar{U}$, $f(z) = 1$ if the first player wins, and $f(z) = 0$ if the first player loses. For two-player Yahtzee, a game may end in a draw. In this case we can set $f(z) = 0.5$. However, it depends on the goal of the two players. If the goal of the first player is either to win or to draw the game, we set $f(z) = 1$ for draw positions. Assuming the zero-sum property holds, the goal of the second player is to win the game. On the other hand, if a draw means nothing different from a loss to the first player, we set $f(z) = 0$ for draw positions.

For each non-terminal roll position $u \in U \cup \bar{U}$, the weight $0 < p((u, v)) \leq 1$ indicates the probability that the game in u will change into move position v.

Therefore,

$$\sum_{\forall v \text{ with } (u,v)\in E} p((u,v)) = 1.$$

Recall that $f(u)$ is the expected score that the first player achieves. For each non-terminal roll position $u \in U \cup \bar{U}$,

$$f(u) = \sum_{\forall v \text{ with } (u,v)\in E} p((u,v))f(v). \tag{1}$$

In optimal play, the first player maximizes f whereas the second player minimizes f. Therefore, for all non-terminal move positions $v \in V \cup \bar{V}$,

$$f(v) = \begin{cases} \max\{f(u) : (v,u) \in E\} \text{ if } v \in V, \\ \min\{f(u) : (v,u) \in E\} \text{ if } v \in \bar{V}. \end{cases} \tag{2}$$

For all positions $w \in U \cup \bar{U} \cup V \cup \bar{V}$, $f(w)$ is also called the *position value* of w.

A database f satisfying both conditions (1) and (2) is called a *solution*. A game is *solved* if a solution is obtained. Unless otherwise noted, all the game graphs in this paper represent finite, zero-sum, two-player, probabilistic games with perfect information, abstracted as above.

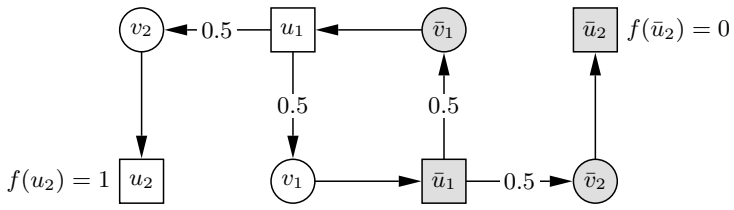

Fig. 1. An example of two-player game graph $G = (U, V, \bar{U}, \bar{V}, E)$.

We illustrate an example in Figure 1, where $u_1, u_2 \in U$, $v_1, v_2 \in V$, $\bar{u}_1, \bar{u}_2 \in \bar{U}$, and $\bar{v}_1, \bar{v}_2 \in \bar{V}$. The two terminal vertices are u_2 and $\bar{u}_2$ with $f(u_2) = 1$ and $f(\bar{u}_2) = 0$, respectively. This example simulates the last stage of a game of two-player Can't Stop. At position u_1, the first player has 50% chance of winning the game immediately, and a 50% chance of being unable to advance and therefore making no progress at this turn. The second player is in the same situation at position $\bar{u}_1$. By (1) and (2),

$$\begin{array}{llll} f(u_1) = \frac{1}{2}f(v_1) + \frac{1}{2}f(v_2), & f(v_1) = f(\bar{u}_1), & f(v_2) = f(u_2) = 1, \\ f(\bar{u}_1) = \frac{1}{2}f(\bar{v}_1) + \frac{1}{2}f(\bar{v}_2), & f(\bar{v}_1) = f(u_1), & f(\bar{v}_2) = f(\bar{u}_2) = 0. \end{array}$$

The unique solution is $f(u_1) = f(\bar{v}_1) = \frac{2}{3}$ and $f(u_2) = f(\bar{v}_2) = \frac{1}{3}$.

The problem of solving a two-player probabilistic game is formulated as follows. Suppose we are given a game graph $G = (U, V, \bar{U}, \bar{V}, E)$ with function

values of the terminal vertices well-defined. First, we investigate the existence and uniqueness of the solution. Second, we design an efficient algorithm to construct the database f, assuming a solution exists.

3 Retrograde Analysis for Two-Player Probabilistic Games

A retrograde algorithm typically consists of three phases: initialization phase, propagation phase, and the final phase. In the initialization phase, the terminal vertices are associated with their position values. In the propagation phase, the information is propagated iteratively back to its predecessors until no propagation is possible. The final phase deals with the undetermined vertices.

Subsection 3.1 gives an algorithm to construct the database of an acyclic game graph. In Subsection 3.2 we prove that two-player Can't Stop has a unique solution. In Subsection 3.3 we develop an algorithm to construct the database for a game graph with cycles.

3.1 Game Graph is Acyclic

For games with acyclic game graphs, such as two-player Yahtzee, the bottom-up propagation procedure is clear. Algorithm 1 gives the pseudocode to construct the database for an acyclic game graph. It can also be applied to constructing the perfect Backgammon bear-off databases with no piece on the bar[2].

Consider Algorithm 1. Assuming all terminal vertices are in $U \cup \bar{U}$, the set S_2 is initially empty and (†) is not required. However, it is useful for the reduced graph $\hat{G}$ in Algorithms 2 and 3. We say a vertex is *determined* if its position value is known. By (1) and (2), a non-terminal vertex cannot be determined until all its children are determined. The sets S_1 and S_2 store all determined but not yet propagated vertices. A vertex is removed from them after it is propagated. The optimal playing strategy is clear: given $v \in V$, always make the move (v, u) with the maximum $f(u)$. For $\bar{v} \in \bar{V}$, we make the move $(\bar{v}, \bar{u})$ with the minimum $f(\bar{u})$. The proof of Lemma 1 reveals that the acyclic property ensures all vertices are determined at the end of propagation phase. Therefore, a final phase is not required.

Lemma 1. *If a game graph is acyclic, its solution exists and is unique.*

Proof. In an acyclic graph, the *level* (the longest distance to the terminal vertices) for each vertex is well-defined. In Algorithm 1, the position values are uniquely determined level by level. Hence the solution exists and is unique. □

[2] To save constructing time and database space, we may use the optimal solution of the simplified one-player game for the approximate Backgammon bear-off databases with no piece on the bar.

Algorithm 1 Construct database f for an acyclic game graph.

Require: $G = (U, V, \bar{U}, \bar{V}, E)$ is acyclic.
Ensure: Program terminates with (1) and (2) satisfied. ▷ Lemma 1
  $\forall u \in U \cup \bar{U}$, $f(u) \leftarrow 0$. ▷ Initialization Phase
  $\forall v \in V$, $f(v) \leftarrow -\infty$, $\forall v \in \bar{V}$, $f(v) \leftarrow \infty$.
  $S_1 \leftarrow \{\text{terminal positions in } U \cup \bar{U}\}$
  $S_2 \leftarrow \{\text{terminal positions in } V \cup \bar{V}\}$ ▷ (†)
  $\forall u \in S_1 \cup S_2$, set $f(u)$ to be its value.
  repeat ▷ Propagation Phase
    for all $u \in S_1$ and $(v, u) \in E$ **do**
      if $v \in V$ **then**
        $f(v) \leftarrow \max\{f(v), f(u)\}$
      else$[v \in \bar{V}]$
        $f(v) \leftarrow \min\{f(v), f(u)\}$
      end if
      if all children of v are determined **then** ▷ (*)
        $S_2 \leftarrow S_2 \cup \{v\}$
      end if
    end for
    $S_1 \leftarrow \emptyset$
    for all $v \in S_2$ and $(u, v) \in E$ **do**
      $f(u) \leftarrow f(u) + p((u, v))f(v)$
      if all children of u are determined **then** ▷ (**)
        $S_1 \leftarrow S_1 \cup \{u\}$
      end if
    end for
    $S_2 \leftarrow \emptyset$
  until $S_1 \cup S_2 = \emptyset$

Note that in Algorithm 1, an edge (u, v) can be visited as many times as the out-degree of u because of (*) and (**). The efficiency can be improved as follows. We associate each vertex with a number of undetermined children, and decrease the value by one whenever a child is determined. A vertex is determined after the number is decreased down to zero. As a result, each edge is visited only once and the algorithm is linear. This is called the *children counting* strategy. For games like Yahtzee, the level of each vertex, the longest distance to the terminal vertices, is known *a priori*. Therefore, we can compute the position values level by level. Each edge is visited only once without counting the children.

3.2 Game Graph is Cyclic

If a game graph is cyclic, a solution may not exist. Even if it exists, it may not be unique. We give a condition under which a solution exists and is unique in Lemma 2. The proof uses the Fixed Point Theorem[3]. With Lemma 2, we

[3] See, e.g., `http://mathworld.wolfram.com/FixedPointTheorem.html`.

prove that the game graph of two-player Can't Stop has a unique solution in Theorem 2.

Theorem 1 (Fixed Point Theorem). *If a continuous function $f : R \longrightarrow R$ satisfies $f(x) \in [a, b]$ for all $x \in [a, b]$, then f has a fixed point in $[a, b]$ (i.e., $f(c) = c$ for some $c \in [a, b]$).*

Lemma 2. *Given a cyclic two-player game graph $G = (U, V, \bar{U}, \bar{V}, E)$, we use G_1 and G_2 to denote the two subgraphs of G induced by $U \cup V$ and $\bar{U} \cup \bar{V}$, respectively. G has a solution with all position values in $[0, 1]$ if,*

1. *The graphs G_1 and G_2 are acyclic.*
2. *There exist $w_1 \in U$ and $w_2 \in \bar{U}$ such that all edges from G_1 to G_2 end at w_2, and all edges from G_2 to G_1 end at w_1. In other words, $E \cap (V \times \bar{U}) \subseteq V \times \{w_2\}$ and $E \cap (\bar{V} \times U) \subseteq \bar{V} \times \{w_1\}$.*
3. *All the terminal position values are in $[0, 1]$.*

In addition, if there is a path in G_1 from w_1 to a terminal vertex $z \in U \cup V$ with position value $f(z) = 1$, then the solution has all position values in $[0, 1]$ and is unique.

Proof. Let $\hat{G}_1 = (U \cup \{w_2\}, V, E_1)$ and $\hat{G}_2 = (\bar{U} \cup \{w_1\}, \bar{V}, E_2)$ be the induced bipartite subgraphs of G. By condition 1, $\hat{G}_1$ and $\hat{G}_2$ are acyclic. Consider $\hat{G}_1$. All the terminal vertices in $\hat{G}_1$ other than w_2 are also terminal in G. If we know the position value of w_2, then by Lemma 1 the solution to $\hat{G}_1$ can be uniquely determined. Let y be the estimated position value of w_2. We can construct a database for $\hat{G}_1$ by Algorithm 1. Denote by $\hat{f}_1(y, w)$ the position value of $w \in U \cup V$ that depends on y. Likewise, given x as the estimated position value of w_1, we denote by $\hat{f}_2(x, \bar{w})$ the position value of $\bar{w} \in \bar{U} \cup \bar{V}$. The values of $\hat{f}_1(y, w)$ for $w \in U \cup V$ and $\hat{f}_2(x, \bar{w})$ for $\bar{w} \in \bar{U} \cup \bar{V}$ constitute a solution to G, if and only if $\hat{f}_1(y, w_1) = 0$ and $\hat{f}_2(x, w_2) = 0$. The main theme of this proof is to discuss the existence and uniqueness of x and y satisfying $\hat{f}_1(y, w_1) = x$ and $\hat{f}_2(x, w_2) = y$, or equivalently $\hat{f}_2(\hat{f}_1(y, w_1), w_2) = y$.

Condition 3 states that all terminal position values are in $[0, 1]$. Iteratively applying (1) and (2), $\hat{f}_2(\hat{f}_1(0, w_1), w_2) \geq 0$ and $\hat{f}_2(\hat{f}_1(1, w_1), w_2) \leq 1$. By Theorem 1, there exists $y^* \in [0, 1]$ such that $\hat{f}_2(\hat{f}_1(y^*, w_1), w_2) = y^*$. Iteratively applying (1) and (2) again, the position values of $w \in U \cup V$, $\hat{f}_1(y^*, w)$, are all in $[0, 1]$. Likewise, the position values of $\bar{w} \in \bar{U} \cup \bar{V}$, $\hat{f}_2(x^*, \bar{w})$, are also all in $[0, 1]$, where $x^* = \hat{f}_1(y^*, w_1)$.

Now we investigate the uniqueness of the solution. Consider $\hat{G}_1$, whose solution can be obtained by propagation that depends on y, the position value of w_2. For convenience of discussion, we propagate in terms of y (i.e., treat y as a variable during the propagation), even though we know the value of y. For example, assuming $y = \frac{1}{2}$, we write $\max\{\frac{2}{3}y, \frac{1}{4}y + \frac{1}{6}\} = \frac{2}{3}y$ instead of $\frac{1}{3}$. Iteratively applying (1) and (2), all propagated values of $u \in U \cup V$ are in the form $ay+b$, which represents the local function values of $\hat{f}_1(y, u)$. By condition 3,

$0 \leq a+b \leq 1$ with a, b nonnegative. We are particularly concerned with $\hat{f}_1(y, w_1)$. Analysis above shows that $\hat{f}_1(y, w_1)$ in value is piecewise linear, continuous and non-decreasing with the slope of each line segment in $[0, 1]$, and so is $\hat{f}_2(x, w_2)$ by a similar discussion. These properties are inherited by the composite function $\hat{f}_2(\hat{f}_1(y, w_1), w_2)$. The additional condition, the existence of a path in G_1 from w_1 to a terminal position with position value 1 further ensures each line segment $ay + b$ of $\hat{f}_1(y, w_1)$ has the slope $a < 1$. Hence the slope of each line segment of $\hat{f}_2(\hat{f}_1(y, w_1), w_2)$ is also less than 1. This guarantees the uniqueness of the solution in $[0, 1]$ to $\hat{f}_2(\hat{f}_1(y, w_1), w_2) = y$. □

Consider the strongly connected components of the game graph of two-player Can't Stop. Each strongly connected component consists of all the positions with a certain placement of the squares and various placement of the at most three markers for each player. The two roll positions with no marker are the *anchors* of the component. In one of them it is the turn of the first player, whereas in the other it is the second player to move. When left without a legal move, the game goes back to one of the two anchors, and results in a cycle. The outgoing edges of each non-terminal component lead to the anchors in the supporting components. The terminal components are those in which some player has won three columns. Each terminal component has only one vertex with position value 1 (if the first player wins) or 0 (if the second player wins).

Theorem 2. *The game graph of two-player Can't Stop has a unique solution, where all the position values are in* $[0, 1]$.

Proof. The proof is by finite induction. We split the graph into strongly connected components, and consider the components in bottom-up order.

Given a non-terminal component with the anchors in its supporting components having position values uniquely determined in $[0, 1]$, we consider the subgraph induced by the component and the anchors in its supporting components. This subgraph satisfies all conditions in Lemma 2, where the terminal positions are the anchors in the supporting components. Therefore, it has a unique solution with all position values in $[0, 1]$. By induction, the solution to the game graph of two-player Can't Stop exists and is unique. □

3.3 Retrograde Approximation Algorithms

If we apply Algorithm 1 to a game graph with cycles, then the vertices in the cycles cannot be determined. A naive algorithm to solve the game is described as follows. Given a cyclic game graph $G = (U, V, \bar{U}, \bar{V}, E)$, we prune some edges so the resulting $\hat{G} = (U, V, \bar{U}, \bar{V}, \hat{E})$ is acyclic, and then solve $\hat{G}$ by Algorithm 1. The solution to $\hat{G}$ is treated as the initial estimation for G, denoted by function $\hat{f}$. We approximate the solution to G by recursively updating $\hat{f}$ using (1) and (2). If $\hat{f}$ converges, it converges to a solution to G. The pseudocode is given in Algorithm 2.

An example is illustrated by solving $G = (U, V, U, \bar{V}, E)$ in Figure 1. We remove $(v_1, \bar{u}_1)$ to obtain the acyclic graph $\hat{G}$, and initialize the newly terminal

Algorithm 2 A naive algorithm to solve a cyclic game graph.

Ensure: If $\hat{f}$ converges, it converges to a solution to $G = (U, V, \bar{U}, \bar{V}, E)$.
  Obtain an acyclic graph $\hat{G} = (U, V, \bar{U}, \bar{V}, \hat{E})$, where $\hat{E} \subset E$. ▷ Estimation Phase
  Compute the solution $\hat{f}$ to $\hat{G}$ by Algorithm 1. ▷ (†)
  Use $\hat{f}$ as the initial guess for G.
  $S_1 \leftarrow \{\text{terminal positions of } \hat{G} \text{ in } U \cup \bar{U}\}$
  $S_2 \leftarrow \{\text{terminal positions of } \hat{G} \text{ in } V \cup \bar{V}\}$
  repeat ▷ Approximation Phase
    for all $u \in S_1$ and $(v, u) \in E$ **do**
      $\hat{f}(v) \leftarrow \begin{cases} \max\{\hat{f}(w) : (v, w) \in E\} \text{ if } v \in V, \\ \min\{\hat{f}(w) : (v, w) \in E\} \text{ if } v \in \bar{V}. \end{cases}$ ▷ (*)
      $S_2 \leftarrow S_2 \cup \{v\}$
    end for
    $S_1 \leftarrow \emptyset$
    for all $v \in S_2$ and $(u, v) \in E$ **do**
      $\hat{f}(u) \leftarrow g(u) + \sum_{\forall w \text{ with } (u,w) \in E} p((u, w))\hat{f}(w)$ ▷ (**)
      $S_1 \leftarrow S_1 \cup \{u\}$
    end for
    $S_2 \leftarrow \emptyset$
  until $\hat{f}$ converges.

vertex u_1 with position value 0. The solution for $\hat{G}$ has $\hat{f}(u_1) = \frac{1}{2}$. The update is repeated with $\hat{f}(u_1) = \frac{5}{8}, \frac{21}{32}, \ldots, \frac{1}{2} + \frac{1}{6}\frac{4^n - 1}{4^n}, \ldots$, which converges to $\frac{2}{3}$. Hence $\hat{f}$ converges to the solution to G. Let e_n be the difference between $\hat{f}(u_1)$ at the nth step and the converged value; then $\frac{e_{n+1}}{e_n} = \frac{1}{4}$. Hence it converges linearly.

Consider Algorithm 2. For two-player Can't Stop, it is natural to prune the outgoing edges of the anchors and obtain the acyclic $\hat{G}$. In (†), we assign an estimated value to each vertex terminal in $\hat{G}$ but not terminal in G (i.e., the newly terminal positions). For efficiency, we do not have to recompute the whole (*) and (**). Updating with the recent changes of the children is sufficient.

If the conditions in Lemma 2 are satisfied (e.g., a strongly connected component of two-player Can't Stop), $\hat{G}$ can be obtained by pruning the outgoing edges of the two anchors w_1 and w_2.

The proof of Lemma 2 reveals that if we solve $\hat{f}_1(y, w_1) = x$ and $\hat{f}_2(x, w_2) = y$ using the 2-dimensional Newton's method[4], then quadratic convergence can be expected. In other words, if we use e_n to denote the difference between the estimation and the solution at the nth step, $\frac{e_{n+1}}{e_n^2} \approx c$ for some constant c when the estimate is close enough to the solution[5]. An example is illustrated with the game graph in Figure 1 as follows. We treat u_1 as w_1 and $\bar{u}_1$ as w_2 in Lemma 2, and let x and y be the initial estimate of the position values of u_1 and $\bar{u}_1$,

[4] See, e.g., `http://www.math.gatech.edu/čarlen/2507/notes/NewtonMethod.html`.

[5] In our case $\hat{f}(x, w)$ is piecewise linear. Hence Newton's method can reach the solution in a finite number of steps. In practice, however, rounding errors may create minor inaccuracy.

respectively. Then $\hat{f}_1(y, u_1) = \frac{1}{2}y + \frac{1}{2}$, $\hat{f}_2(x, \bar{u}_1) = \frac{1}{2}x$. Solving $\frac{1}{2}y + \frac{1}{2} = x$ and $\frac{1}{2}x = y$, we obtain $x = \frac{2}{3}$ and $y = \frac{1}{3}$, which are the exact position values of u_1 and $\bar{u}_1$, respectively. In this small example we obtain the solution by one iteration. In practice, multiple iterations are expected to reach the solution. The pseudocode is given in Algorithm 3.

Algorithm 3 An efficient algorithm to solve a cyclic game graph.

Require: $G = (U, V, \bar{U}, \bar{V}, E)$ satisfies the conditions in Lemma 2.
Ensure: $\hat{f}_1$ and $\hat{f}_2$ converge to a solution to G in the rate of Newton's method.
{Estimation Phase:}
Denote the induced subgraphs $\hat{G}_1 = (U \cup \{w_2\}, V, E_1)$ and $\hat{G}_2 = (\bar{U} \cup \{w_1\}, \bar{V}, E_2)$.
Assuming the conditions in Lemma 2 hold, $\hat{G}_1$ and $\hat{G}_2$ are acyclic and $E_1 \cup E_2 = E$.
Estimate the position values of anchors $w_1 \in U$ and $w_2 \in \bar{U}$, denoted by x and y.
{Approximation Phase:}
repeat
  Solve $\hat{G}_1$ in terms of the current estimate y for w_2 by Algorithm 1. ▷ (*)
  Denote the linear segment of $\hat{f}_1(y, w_1)$ by $a_1 y + b_1$.
  Solve $\hat{G}_2$ in terms of the current estimate x for w_1 by Algorithm 1. ▷ (**)
  Denote the linear segment of $\hat{f}_2(x, w_2)$ by $a_2 x + b_2$.
  Solve $x = a_1 y + b_1$ and $y = a_2 x + b_2$ for the next estimates x and y.
until the values of x and y cannot be longer unchanged.

Consider Algorithm 3. In the estimation phase, the better the initial estimated position values of w_1 and w_2 (denoted by x and y respectively), the fewer steps are needed to reach the solution. In the approximation phase, the graphs $\hat{G}_1$ and $\hat{G}_2$ are disjoint except w_1 and w_2, and the propagations in (*) and (**) in each iteration are independent of each other. Therefore, Algorithm 3 is natively parallel on two processors, by separating the computations (*) and (**).

A more general model is that a game graph G has two anchors w_1, w_2 (i.e., removing the outgoing edges of w_1 and w_2 results in an acyclic graph), but the precondition in Lemma 2 does not hold. In this model the incorporation of 2-dimensional Newton's method is still possible as follows. Let x and y be the current estimated position values of w_1 and w_2 at each iteration, respectively. The propagated values of w_1 and w_2 in terms of x and y (e.g., if $x = \frac{1}{2}$ and $y = \frac{3}{4}$, we write $\min\{\frac{1}{2}x + \frac{1}{2}y, \frac{2}{3}x + \frac{1}{3}y\} = \frac{2}{3}x + \frac{1}{3}y$ instead of $\frac{7}{12}$) are denoted by $\hat{f}(x, y, w_1)$ and $\hat{f}(x, y, w_2)$. We solve the linear system of $\hat{f}(x, y, w_1) = x$ and $\hat{f}(x, y, w_2) = y$ for x and y as the position values of w_1 and w_2 in the next iteration. Three observations are worth noting. First, since the precondition in Lemma 2 does not hold, existence and uniqueness of the solution requires further investigation. Second, the algorithm is not natively parallel on two processors as stated above. Third, this generalization relies on the property of two anchors, not two players. It also applies to a one-player probabilistic game graph with two anchors.

4 Indexing Scheme

We use two different indexing schemes for Can't Stop positions, one scheme for anchors and another for non-anchors. The indexing scheme for non-anchor positions is designed so that, given an index we can quickly compute the positions of all of the markers, and vice versa. It is a mixed radix system in which there is a digit for each player and column representing the position of that player's marker in the column; this scheme is similar to that used for one-player Can't Stop [5]. A different scheme is used for anchors so that we can store the position value database in a compact form.

In the variant used in our experiments, an anchor $(x_2, ..., x_{12}, y_2, ..., y_{12}, t)$ is illegal if $x_i = y_i \neq 0$ for some i (players' markers cannot occupy the same location with a column). With this restriction many indices map to illegal anchors. Furthermore, once a column is closed, the locations of the markers in that column are irrelevant; only which player won matters. For example, if $y_2 = 3$ then we can set $x_2 = 0$ and the resulting position represents the position where $x_2 \in \{1, 2\}$ as well. If the position values are stored in an array indexed using the mixed radix system as for non-anchors, then the array would be sparse: for the official game about 98% of the entries would be wasted on illegal and equivalent indices.

In order to avoid wasting space in the array and to avoid the structural overhead needed for more advanced data structures, a different indexing scheme is used that results in fewer indices mapping to illegal, unreachable, or equivalent positions.

Write each position as $((x_2, y_2), ..., (x_{12}, y_{12}), t)$. Associate with each pair (x_i, y_i) an index z_i corresponding to its position on a list of the legal pairs of locations in column i (that is, on a list of ordered pairs (x, y) such that $x \neq y$). The z_i and t are then used as digits in a mixed radix system to obtain the index

$$t + \sum_{c=2}^{12} z_c \cdot 2 \prod_{d=2}^{c-1} (3 + l_d(l_d - 1))$$

where l_d is the length of column d and the term in the product is the number of legal, distinct pairs of locations in column d. The list of ordered pairs used to define the z_i's can be constructed so that if component u is a supporting component of v then the indices of u's anchors are greater than the indices of v's and therefore we may iterate through the components in order of decreasing index to avoid counting children while computing the solution.

There is still redundancy in this scheme: when multiple columns are closed, what is important is which columns have been closed and the total number of columns won by each player, but not which player has won which columns. Before executing Algorithm 1 on a component, we check whether an equivalent component has already been solved. We deal with symmetric positions in the same way.

5 Experiments

As proof of concept, we have solved simplified versions of two-player Can't Stop. The simplified games use dice with fewer than six sides and may have shorter columns than the official version. Let (n, k) Can't Stop denote the two-player game played with n-sided dice and columns of length $k, k+2, ..., k+2(n-1), ..., k$.

We have implemented Algorithm 3 in Java and solved $(3, k)$ Can't Stop for $k = 1, 2, 3$. We used an initial estimate of $(\frac{1}{2}, \frac{1}{2})$ for the position values of the anchors within a component. Table 1 shows, for three versions of the game, the size of the game graph, the time it took the algorithm to run, and the probability that the first player wins assuming that each player plays optimally. The listed totals for components and positions within those components excludes the components not examined because of equivalence.

Table 1. Results of solving simple versions of Can't Stop.

(n, k)	Components	Total positions	Time	P(P1 wins)
$(3, 1)$	6,324	634,756	4m33s	0.760
$(3, 2)$	83,964	20,834,282	3h45m	0.711
$(3, 3)$	930,756	453,310,692	3d13h	0.689

Note that the time to solve the game grows faster than the number of positions. This is because the running time is also dependent on the number of iterations per component, which is related to the quality of the initial estimate and the complexity of the game. Table 2 gives the average number of iterations versus the position value of the component, given as (x, y) where x (y) is the probability that the first player wins given that the game has entered the component and it is the first (second) player's turn. Note that the table is upper triangular because there is never an advantage in losing one's turn and symmetric because of symmetric positions within the game. Perhaps surprisingly, the components that require the most iterations are not those where the solution is farthest from the initial estimate of $(\frac{1}{2}, \frac{1}{2})$. We conjecture that this is because positions where there is a large penalty for losing one's turn require less strategy (the decision will usually be be to keep rolling) and therefore $\hat{f}$ is less complicated (has fewer pieces) and so Newton's method converges faster.

6 Conclusion

We used a four-partite graph to abstract a two-player probabilistic game. Given a position, its position value indicates the winning rate of the first player in optimal play. We investigated the game of two-player Can't Stop, and proved that its optimal solution exists and is unique. To obtain the optimal solution,

Table 2. Iterations required vs. position values for (3, 3) Can't Stop

		x				
		0.0-0.2	0.2-0.4	0.4-0.6	0.6-0.8	0.8-1.0
	0.0-0.2	3.00	3.25	3.50	3.37	2.87
	0.2-0.4	-	3.27	4.12	3.83	3.37
y	0.4-0.6	-	-	2.50	4.12	3.50
	0.6-0.8	-	-	-	3.27	3.25
	0.8-1.0	-	-	-	-	3.00

we generalized an approximation algorithm from [5] by incorporating the 2-dimensional Newton's method with retrograde analysis. The technique was then used to solve simplified versions of two-player Can't Stop. The official version has over 10^{36} components – too many to solve with currently available technology. It may be possible to find patterns in the solutions to the simplified games and use those patterns to approximate optimal solutions to the official game.

References

1. H.-r. Fang. The nature of retrograde analysis for Chinese chess, part I. *ICGA Journal*, 28(2):91–105, 2005.
2. H.-r. Fang. The nature of retrograde analysis for Chinese chess, part II. *ICGA Journal*, 28(3):140–152, 2005.
3. R. Gasser. Solving Nine Men's Morris. *Computational Intelligence*, 12:24–41, 1996.
4. J. Glenn. An optimal strategy for Yahtzee. Technical Report CS-TR-0002, Loyola College, 4501 N. Charles St, Baltimore MD 21210, USA, May 2006.
5. J. Glenn, H.-r. Fang, and C. P. Kruskal. A retrograde approximate algorithm for one-player can't stop. Accepted by CG'06 conference, to appear, 2006.
6. G. Irving, J. Donkers, and J. Uiterwijk. Solving Kalah. *ICGA Journal*, 23(3):139–147, 2000.
7. R. Lake, J. Schaeffer, and P. Lu. Solving large retrograde analysis problems using a network of workstations. In H.J. van den Herik, I. S. Herschberg, and J.W.H.M. Uiterwijk, editors, *Advances in Computer Games VII*, pages 135–162. University of Limburg, Maastricht. the Netherlands, 1994.
8. J.W. Romein and H.E. Bal. Solving the game of Awari using parallel retrograde analysis. *IEEE Computer*, 36(10):26–33, October 2003.
9. J. Schaeffer, Y. Björnsson, N. Burch, R. Lake, P. Lu, and S. Sutphen. Building the checkers 10-piece endgame databases. In H.J. van den Herik, H. Iida, and E.A. Heinz, editors, *Advances in Computer Games 10. Many Games, Many Challenges*, pages 193–210. Kluwer Academic Publishers, Boston, USA, 2004.
10. K. Thompson. Retrograde analysis of certain endgames. *ICCA Journal*, 9(3):131–139, 1986.
11. K. Thompson. 6-piece endgames. *ICCA Journal*, 19(4):215–226, 1996.
12. P. Woodward. Yahtzee: The solution. *Chance*, 16(1):18–22, 2003.
13. R. Wu and D.F. Beal. Fast, memory-efficient retrograde algorithms. *ICGA Journal*, 24(3):147–159, 2001.