A Retrograde Approximation Algorithm for Multi-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, MD 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. An n-player, finite, probabilistic game with perfect information can be presented as a $2n$-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. We have presented our success on tackling one-player Can't Stop and two-player Can't Stop. In this article we study the computational solution of multi-player Can't Stop (more than two players), and present a retrograde approximation algorithm to solve it by incorporating the multi-dimensional Newton's method with retrograde analysis. Results of experiments on small versions of three- and four-player Can't Stop are presented.

1 Introduction

Retrograde analysis has been successfully applied to convergent, deterministic, finite, and two-player zero-sum games with perfect information [13], such as checkers [12] and Awari [11]. In contrast, its application to probabilistic games was generally limited to those with game graph representation being acyclic, such as Yahtzee [6, 14] and Solitaire Yahtzee [7]; Pig[4] is a notable exception [10]. We consider the probabilistic games in graph representation with cycles, and are particularly interested in Can't Stop[5]. Our success of tackling one-player and two-player Can't Stop was presented in [8] and [9], respectively. This article

[4] See, e.g., http://en.wikipedia.org/wiki/Pig_(dice).

[5] Can't Stop was designed by Sid Sackson and marketed first by Parker Brothers and now by Face 2 Face Games. It has won a Major Fun award from Majorfun.com and received a Preferred Choice Award from Creative Child Magazine. The rules can be found at http://en.wikipedia.org/wiki/Can't_Stop.

presents our study of multi-player Can't Stop that allows more than two players. Our method can also be applied to the multi-player versions of some other probabilistic games, such as Pig, Pig Mania[6], and Hog[7].

An n-player probabilistic game can be represented as a $2n$-partite graph $G = (U_1, \ldots, U_n, V_1, \ldots, V_n, E)$, where U_i corresponds to random events and V_i corresponds to deterministic events for the ith players for $i = 1, \ldots, n$, and $E = (\bigcup_{i=1}^{n}(U_i \times V_i)) \cup (\bigcup_{i=1}^{n} V_i \times \bigcup_{i=1}^{n} U_i)$. 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 give a retrograde approximation algorithm to solve n-player Can't Stop, by incorporating the n-dimensional Newton's method into a retrograde algorithm. This indeed is a generalization of the method for two-player Can't Stop [9, 5].

The rest of this paper is organized as follows. Section 2 abstracts multi-player probabilistic games. Section 3 gives a retrograde approximation algorithm to solve multi-player Can't Stop. Section 4 presents the indexing scheme. Section 5 summarizes the results of the experimental tests. Our findings are summarized in Sect. 6.

2 Abstraction of Probabilistic Games

We use a game graph $G = (U_1, \ldots, U_n, V_1, \ldots, V_n, E)$ to represent an n-player probabilistic game ($n \geq 2$), where roll and move positions of the ith player are in U_i and V_i, respectively, for $i = 1, \ldots, n$, and $E = (\bigcup_{i=1}^{n} U_i \times V_i) \cup (\bigcup_{i=1}^{n} V_i \times \bigcup_{i=1}^{n} U_i)$. Each position u is associated with a vector of scores $f(u) = (f_1(u), \ldots, f_n(u)) \in \mathbb{R}^n$, where $f_i(u)$ represents the expected score that the ith player achieves in optimal play from u for $i = 1, \ldots, n$. This mapping is denoted by a function $f : \bigcup_{i=1}^{n} U_i \cup V_i \to \mathbb{R}^n$, which is also called a *database* of the game.

For each non-terminal roll position $u \in \bigcup_{i=1}^{n} U_i$, each outgoing edge (u, v) has a weight $0 < p((u, v)) \leq 1$ indicating the probability that the game in u will change into move position v. Then

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

In optimal play, each player maximizes locally his score[8]. Consider the move positions in V_i of the ith player. For all non-terminal move positions $v_i \in V_i$,

$$f(v_i) = f(\text{argmax}\{f_i(u) : (v_i, u) \in E\}). \tag{2}$$

In other words, the ith player chooses the move to maximize his score at each move position $v_i \in V_i$. A database f that satisfies both conditions (1) and (2) is called a *solution* to G.

[6] See, e.g., `http://en.wikipedia.org/wiki/Pass_the_Pigs`.
[7] See, e.g., `http://en.wikipedia.org/wiki/Pig_(dice)#Rule_Variations`.
[8] We use 'he/his' when both 'she/her' and 'he/his' are possible, respectively.

First, we consider (1). In this paper we let $f_i(u) \in [0,1]$ be the probability that the ith player at position u will win the game in optimal play, although in general it can be from any scoring method. Note that we have no assumption of the number of winners at the end of a game. It can be no winner or multiple winners. If a game always ends with exactly one winner, then $\sum_{i=1}^{n} f_i(w) = 1$ for all $w \in \bigcup_{i=1}^{n} U_i \cup V_i$. If in addition $n = 2$, this model coincides with the zero-sum two-player model presented in [9, 5] by setting $f_2(u) = 1 - f_1(u)$.

Then we consider (2). Ambiguity occurs if there is more than one maximizer of $f_i(u)$ that results in a different $f(u)$. In such a case two possible disambiguation rules are listed as follows.

- Take the average of $f(u)$ of all maximizers of $f_i(u)$, which means to choose randomly a maximizer.
- Choose a maximizer according to additional assumptions (e.g., collusion between players).

If some ith player's goal is not to maximize his own score but to attack some jth player, then (2) is replaced by

$$f(v_i) = f(\text{argmin}\{f_j(u) : (v_i, u) \in E\})$$

for $v_i \in V_i$. Ambiguity, if present, can be handled in a similar way stated above.

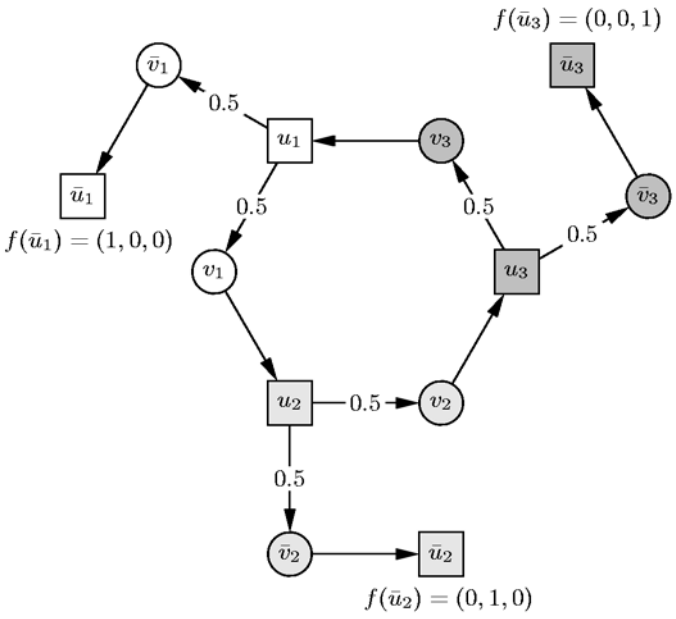

Fig. 1. An example of three-player game graph $G = (U_1, V_1, U_2, V_2, U_3, V_3, E)$

We illustrate an example in Fig. 1, where $u_1, \bar{u}_1 \in U_1$, $v_1, \bar{v}_1 \in V_1$, $u_2, \bar{u}_2 \in U_2$, $v_2, \bar{v}_2 \in V_2$, $u_3, \bar{u}_3 \in U_3$, and $v_3, \bar{v}_3 \in V_3$. The three terminal vertices are $\bar{u}_1$, $\bar{u}_2$ and $\bar{u}_3$ with position values $f(\bar{u}_1) = (1, 0, 0)$, $f(\bar{u}_2) = (0, 1, 0)$, and $f(\bar{u}_3) = (0, 0, 1)$. A cycle is formed by the edges between vertices $u_1, v_1, u_2, v_2, u_3, v_3$. This example simulates the last stage of a game of three-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 and third players are in the same situation at position u_2 and u_3, respectively. Let $f(u_1) = (x, y, z)$. By (1) and (2),

$$\begin{aligned} f(v_2) = f(u_3) &= (\frac{1}{2}x, \frac{1}{2}y, \frac{1}{2}z + \frac{1}{2}), \\ f(v_1) = f(u_2) &= (\frac{1}{4}x, \frac{1}{4}y + \frac{1}{2}, \frac{1}{4}z + \frac{1}{4}), \\ f(v_3) = f(u_1) &= (\frac{1}{8}x + \frac{1}{2}, \frac{1}{8}y + \frac{1}{4}, \frac{1}{8}z + \frac{1}{8}) = (x, y, z). \end{aligned} \tag{3}$$

Solving the last equation in (3), we obtain $x = \frac{4}{7}$, $y = \frac{2}{7}$ and $z = \frac{1}{7}$, the winning probabilities of the three players when it is the first player's turn to move[9]. This example reveals that solving multi-player Can't Stop is equivalent to solving a system of piecewise linear equations. We give in Sect. 3 an approximation algorithm to solve it by incorporating the multi-dimensional Newton's method with retrograde analysis.

Because of the potential ambiguity of (2), the existence and uniqueness of the solution of multi-player Can't Stop may need further assumptions, and hence are not investigated in this paper.

Note that, however, if the number of players is two (i.e., $n = 2$) and the position value $f(u) = (f(u_1), f(u_2))$ satisfies $f_1(u) + f_2(u) = 1$ for $u \in \bigcup_{i=1}^{2} U_i \cup V_i$ (i.e., always exactly one winner at the end), then no ambiguity of (2) would occur. For two-player Can't Stop we have proved the existence and uniqueness of the solution in [5].

3 Retrograde Solution for Multi-Player Can't Stop

Can't Stop is a game for up to four players. It can be generalized to allow even more players. In this section we give a retrograde approximation algorithm for n-player Can't Stop, by incorporating the n-dimensional Newton's method with retrograde analysis. This is generalized from the result of the two-player version game in [9]. We begin with acyclic game graphs for simplicity.

3.1 Game Graph is Acyclic

For games with acyclic game graphs, such as multi-player Yahtzee, the bottom-up propagation procedure is clear. Algorithm 1 gives the pseudocode to construct the database for an acyclic game graph.

[9] Another small example of simplified Parcheesi can be found in [3, Chapter 3].

Algorithm 1 Construct database f for an acyclic game graph

Require: $G = (U_1, V_1, \ldots, U_n, V_n, E)$ is acyclic.
Ensure: Program terminates with (1) and (2) satisfied.
  $\forall u \in \bigcup_{i=1}^{n} U_i$, $f(u) \leftarrow 0^{(n)}$. ▷ Initialization Phase
  $\forall v \in \bigcup_{i=1}^{n} V_i$, $f(v) \leftarrow -\infty^{(n)}$.
  $S_1 \leftarrow \{$terminal positions in $\bigcup_{i=1}^{n} U_i\}$
  $S_2 \leftarrow \{$terminal positions in $\bigcup_{i=1}^{n} V_i\}$ ▷ (†)
  $\forall w \in S_1 \cup S_2$, set $f(w)$ to be its position value.
  repeat ▷ Propagation Phase
    for all $u \in S_1$ and $(v, u) \in E$ **do**
      Determine i such that $v \in V_i$.
      $f(v) \leftarrow f(\text{argmax}\{f_i(w) : (v, w) \in E\})$
      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 $S_1 \cup S_2 = \emptyset$

In Algorithm 1, $0^{(n)}$ means a zero vector of size n, and $-\infty^{(n)}$ follows the same syntax. Assuming all terminal vertices are in $\bigcup_{i=1}^{n} U_i$, 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 $v \in V_i$, a position with the ith player to move, always make the move (v, u) that maximizes $f_i(u)$.

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 a solution exists. A non-terminal position has its value determined whenever all its children are determined at (*) and (**) in Algorithm 1. One can use a boolean array to trace the determined positions in the implementation. The uniqueness of the solution is subject to the disambiguation rule for (2). If all terminal positions w satisfy $\sum_{i=1}^{n} f_i(w) = 1$, so do all the positions, by recursively applying (1) and (2).

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. Note that Algorithm 1, is related to dynamic programming. See, for example, [1] for more information.

3.2 Game Graph is Cyclic

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 $G = (U_1, V_1, \ldots, U_n, V_n, E)$, we prune some edges so the resulting $\hat{G} = (U_1, V_1, \ldots, U_n, V_n, \hat{E})$ 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.

Algorithm 2 A naive algorithm to solve a cyclic game graph

Ensure: If $\hat{f}$ converges, it converges to a solution to $G = (U_1, V_1, \ldots, U_n, V_n, E)$.
  Obtain an acyclic graph $G = (U_1, V_1, \ldots, U_n, V_n, \hat{E})$, $\hat{E} \subset E$. ▷ Estimation Phase
  Compute the solution $\hat{f}$ to $\hat{G}$ by Algorithm 1. ▷ (†)
  Use $\hat{f}$ as the initial guess for G.
  $S_1 \leftarrow \{$terminal positions of $\hat{G}$ in $\bigcup_{i=1}^{n} U_i\}$.
  $S_2 \leftarrow \{$terminal positions of $\hat{G}$ in $\bigcup_{i=1}^{n} V_i\}$.
  repeat ▷ Approximation Phase
    for all $u \in S_1$ and $(v, u) \in E$ **do**
      Determine i such that $v \in V_i$.
      $\hat{f}(v) \leftarrow f(\text{argmax}\{\hat{f}_i(w) : (v, w) \in E\})$. ▷ (*)
    end for
    $S_1 \leftarrow \emptyset$
    for all $v \in S_2$ and $(u, v) \in E$ **do**
      $\hat{f}(u) \leftarrow \sum_{\forall w \text{ with } (u,w)\in E} p((u, w))\hat{f}(w)$ ▷ (**)
      $S_1 \leftarrow S_1 \cup \{u\}$
    end for
    $S_2 \leftarrow \emptyset$
  until $\hat{f}$ converges.

An example is illustrated by solving the game graph in Fig. 1. We remove (v_3, u_1) to obtain the acyclic graph $\hat{G}$, and initialize the newly terminal vertex v_3 with position value $(1, 0, 0)$. The solution for $\hat{G}$ has $\hat{f}(u_1) = (\frac{5}{8}, \frac{1}{4}, \frac{1}{8})$. The update is repeated with $\hat{f}(u_1) = (\frac{5}{8}, \frac{1}{4}, \frac{1}{8}), (\frac{37}{64}, \frac{9}{32}, \frac{9}{32}), \ldots, (\frac{4 \cdot 8^k + 3}{7 \cdot 8^k}, \frac{2(8^k - 1)}{7 \cdot 8^k}, \frac{8^k - 1}{7 \cdot 8^k}), \ldots,$

which converges to $(\frac{4}{7}, \frac{2}{7}, \frac{1}{7})$, the correct position value of u_1. Therefore $\hat{f}$ converges to the solution to G. Let $e_i(k)$ be the magnitude difference between $\hat{f}_i(u_1)$ at the kth step and its converged value; then $\frac{e_i(k+1)}{e_i(k)} = \frac{1}{8}$ for $i = 1, 2, 3$. Hence it converges linearly. This naive Algorithm 2 is related to value iteration. See, for example, [2] for more information.

For computational efficiency, we split a given game graph into strongly connected components, and consider the components in bottom-up order. For multi-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 neutral markers for the player on the move. The roll positions with no marker are the *anchors* of the component. When left without a legal move, the game goes back to one of the 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 in the form $(0, \ldots, 0, 1, 0, \ldots, 0)$; the ith entry is 1 if the i player wins, and otherwise it is 0.

Denote by a cyclic game graph $G = (U_1, V_1, \ldots, U_n, V_n, E)$ a non-terminal component of multi-player Can't Stop and its outgoing edges to the supporting components. Let G_i be the subgraph of G induced by $U_i \cup V_i$ for $i = 1, \ldots, n$. The following two properties hold.

P1 All the graphs G_i for $i = 1, \ldots, n$ are acyclic.

P2 There exist $w_i \in U_i$, such that the edges from G_i to the other vertices (i.e., not in $U_i \cup V_i$) all end at w_{i+1} for $i = 1, \ldots, n$, where we have defined $w_{n+1} \equiv w_1$ for notational convenience.

Properties (P1) and (P2) also hold in some other probabilistic games in strongly connected components, such as Pig, Pig Mania, and Hog. Therefore, the following discussion and our method are applicable to these games.

Let $\hat{G}_i = (U_i \cup \{w_{i+1}\}, V_i, E_i)$ be the induced bipartite subgraph of G for $i = 1, \ldots, n$. By property (P1), $\hat{G}_i$ is acyclic. All the terminal vertices in $\hat{G}_i$ other than w_{i+1} are also terminal in G. By property (P2), the union of $\hat{G}_i$ for $i = 1, \ldots, n$ forms G. Let x_{i+1} be the estimated position value of w_{i+1}. Here $x_{n+1} \equiv x_1$ because of notational convenience $w_{n+1} \equiv w_1$. We can construct a database for $\hat{G}_i$ with x_{i+1} by Algorithm 1. Denote by $\hat{g}_i(x_{i+1}, w)$ the position value of $w \in U_i \cup V_i$ that depends on x_{i+1}. Given $x_2, \ldots, x_{n+1}$, the values of $\hat{g}_i(x_{i+1}, w)$ for $w \in U_i \cup V_i$, $i = 1, \ldots, n$ constitute a solution to G, if and only if

$$\hat{g}_i(x_{i+1}, w_i) = x_i, \qquad i = 1, \ldots, n. \tag{4}$$

The discussion above suggests to solve the system of equations (4) directly. An example is illustrated with the game graph in Fig. 1 as follows. We treat u_1, u_2, u_3 as the three anchors w_1, w_2, w_3, and let x_1, x_2, x_3 be the initial estimate of the position values of them, respectively. The equations (4) are

$$x_3 = 0.5x_1 + (0, 0, 0.5);$$

$$x_2 = 0.5x_3 + (0, 0.5, 0); \tag{5}$$
$$x_1 = 0.5x_2 + (0.5, 0, 0).$$

The solution of this linear system is $x_1 = (\frac{4}{7}, \frac{2}{7}, \frac{1}{7})$, $x_2 = (\frac{1}{7}, \frac{4}{7}, \frac{2}{7})$ and $x_3 = (\frac{2}{7}, \frac{1}{7}, \frac{4}{7})$, which are the exact position values of u_1, u_2 and u_3, respectively.

Solving (4) by propagation in value corresponds to the fixed point iteration for computing fixed points of functions[10], and therefore linear convergence can be expected. In contrast, solution (3) inspires us to propagate in terms of the position value x_1 of w_1. The resulting method corresponds to the n-dimensional Newton's method (see, e.g., [4, Chapter 5]). The pseudocode is given in Algorithm 3.

Algorithm 3 An efficient algorithm to solve a cyclic game graph

Require: $G = (U_1, V_1, \ldots, U_n, V_n, E)$ satisfies properties (P1) and (P2).
Ensure: If $\hat{f}$ converges, it converges to a solution to G in the rate of Newton's method.
{Estimation Phase:}
Denote the induced subgraphs $\hat{G}_i = (U_i \cup \{w_{i+1}\}, V_i, E_i)$ for $i = 1, \ldots, n$; $w_{n+1} \equiv w_1$.
{Note that all $\hat{G}_i$ are acyclic and $\bigcup_{i=1}^{n} E_i = E$}.
Estimate the position values of anchors $w_i \in U_i$, denoted by x_i for $i = 1, \ldots, n$.
{Approximation Phase:}
Estimate the position value of w_{n+1} (i.e., w_1); denote it by x_1.
repeat
  for all $i = n, n-1 \ldots, 1$ **do**
    Solve $\hat{G}_i$ based on x_{i+1} in terms of x_1 by Algorithm 1.
    {Propagation by (1) and (2) is done in terms of x_1.}
  end for
  {We have the position value of w_1 in $\hat{G}_1$ in terms of x_1, denoted by $h(x_1)$.}
  Solve $h(x_1) = x_1$ for new estimate x_1.
until it converges (i.e., x_1 is unchanged in value).

Note that Newton's method needs only one iteration to solve a linear system. Indeed, solution (3) is an illustration of applying Algorithm 3 to solve the small example (5), where (x, y, z) in (3) plays the role of x_1 in Algorithm 3. In this case we obtain the solution by one iteration, since the system (5) is linear. In practice, however, the equations (4) are piecewise linear. Hence, multiple iterations are expected to reach the solution. In the experiments on simplified versions of three- and four-player Can't Stop, it always converged, although Newton's method does not guarantee convergence in general.

Consider Algorithm 3. In the estimation phase, the better the initial estimated position value x_1 of the anchors $w_1 (\equiv w_{n+1})$, the fewer iterations are needed to reach the solution. Assuming the game always has exactly one winner at the end, we may reduce one dimension by setting $f_n(u) = 1 - \sum_{i=1}^{n-1} f_i(u)$ for all positions $u \in \bigcup_{i=1}^{n} U_i \cup V_i$. This change of Algorithm 3 results in a method corresponding to the $(n-1)$-dimensional Newton's method.

[10] See, for example, `http://en.wikipedia.org/wiki/Fixed_point_iteration`.

In Algorithm 3, the graphs $\hat{G}_1, \ldots, \hat{G}_n$ are disjoint except for the anchors $w_1, \ldots, w_n$. Therefore, in the estimation phase, we may initialize the position values $x_1, \ldots, x_n \in \mathbb{R}^n$ of $w_1, \ldots, w_n$. In the approximation phase, we propagate for $\hat{G}_1, \ldots, \hat{G}_n$ in terms of $x_2, \ldots, x_{n+1} \in \mathbb{R}^n$ ($x_{n+1} \equiv x_1$), respectively and separately. The resulting algorithm corresponds to the n^2-dimensional Newton's method and is natively parallel on n processors. We illustrate an example by solving the game graph in Fig. 1. The first Newton's step results in the linear system (5) of 9 variables, which leads to the solution of the game graph in one iteration. (Note that here $x_1, x_2, x_3 \in \mathbb{R}^3$.) In general, the equations (4) are piecewise linear and require multiple Newton's iterations to reach the solution.

A more general model is that an n-player game graph G has m anchors $w_1, \ldots, w_m$ (i.e., removing the outgoing edges of $w_1, \ldots, w_m$ results in an acyclic graph), but does not satisfy properties (P1) and (P2). In this model the incorporation of mn-dimensional Newton's method is still possible, but it may not be natively parallel.

4 Indexing Scheme

We use two different indexing schemes for positions in n-player Can't Stop: one for anchors and another for non-anchors. Because we can discard the position values of non-anchors once we have computed the position value of their anchors, speed is more important than space when computing the indices of non-anchors. Therefore, we use a mixed radix scheme like the one used for one-player Can't Stop [8] and two-player Can't Stop [9] for non-anchor positions.

In this scheme, anchors are described by $(x_2^1, \ldots, x_{12}^1, x_2^2, \ldots, x_{12}^n, t)$ where x_c^p represents the position of player p's square in column c, and t is whose turn it is. Components and positions within components are described in the same way, except that since a component includes n anchors that differ only in whose turn it is, t may be omitted when describing a component, and within a component we record the positions of neutral markers as if they are player t's squares (given the tuple describing a component and a tuple describing a position with that component, we can easily determine where the neutral markers are). Therefore, a position within a component $(x_2^1, \ldots, x_{12}^1, x_2^2, \ldots, x_{12}^n, t)$ is $(y_2^1, \ldots, y_{12}^1, y_2^2, \ldots, y_{12}^n, t)$ where, for all c and p, $y_c^p = x_c^p$, except that in at most three locations we may have $y_c^t > x_c^t$ (player t may have advanced the neutral markers in three columns). The y_c^p and t are used as the digits in the mixed radix system. The place value of the t digit is 1. The place value of the y_2^1 digit is $v_2^1 = 2$, and in general $v_c^p = v_{c-1}^p \cdot (1 + l_{c-1})$ if $c > 2$ and $v_c^p = v_{12}^{p-1} \cdot (1 + l_{12})$ if $c = 2$ and $p > 1$, where l_c denotes the length of column c. The index of a position is then $(t-1) + \sum_{p=1}^{n} \sum_{c=2}^{12} (y_c^p \cdot v_c^p)$.

Because the probability database for the anchors is kept, space is an important consideration when indexing anchors. In the variant used in our experiments, an anchor $(x_2^1, \ldots, x_{12}^n, t)$ is illegal if $x_c^p = x_c^{p'} > 0$ for some $p \neq p'$ (players' squares cannot occupy the same location with a column). Because some positions are not valid anchors, the mixed radix system described above does not define

a bijection between anchors and any prefix of $\mathbb{N}$. The indexing scheme therefore maps some indices to nonexistent anchors.

Furthermore, once a column is closed, the locations of the markers in that column are irrelevant; only which player won matters. For example, an anchor u with $x_5^1 = 9$ and $x_5^2 = x_5^3 = 0$ also represents the positions with $x_5^2, x_5^3 \in \{1, \ldots, 8\}$ and all other markers in the same places as u. If the probability database is stored in an array indexed using the mixed radix system as for non-anchors, then the array would be sparse: for the official 3-player game, over 99.9% 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.

We write each position as $((x_2^1, \ldots, x_2^n), \ldots, (x_{12}^1, \ldots, x_{12}^n), t)$. Associate with each n-tuple $(x_c^1, \ldots, x_c^n)$ an index z_c corresponding to its position on a list of the legal n-tuples of locations in column c (i.e., on a list of n-tuples $(y_c^1, \ldots, y_c^n)$ such that $y_c^i \neq y_c^j$ unless $y_c^i = y_c^j = 0$ or $i = j$, and if $y_c^i = l_c$ then $y_c^j = 0$ for $j \neq i$). For the three-player games and a column with $l_c = 2$ this list would be $(0,0,0), (0,0,1), (0,1,0), (1,0,0), (0,0,2), (0,2,0), (2,0,0)$. Therefore, the anchor $((0,0,1), (2,0,0), (1,0,0), (0,0,0), \ldots, (0,0,0), 1)$ (that is, the position in which player 3 has a square in the first space of column 2, player 1 has squares two spaces into column 3 and one space into column 4, and it is player one's turn) would be encoded as $(1, 6, 3, 0, \ldots, 0, 1)$. Those z_c and t are then used as digits in a mixed radix system to obtain the index

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

where the $T(d)$ term in the product is the number of legal, distinct tuples of locations in column d; $T(d) = n + \sum_{z=0}^{n} \binom{n}{z} P(l_d - 1, z)$. The list of n-tuples 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 columns are closed, what is important is which columns have been closed and the total number won by each player, but not which columns were won by each player. Before executing Algorithm 3 on a component, we check whether an equivalent component has already been solved. We deal with symmetric positions in the same way.

5 Experiments

The official version of three- and four-player Can't Stop has over 10^{25} and 10^{32} components – too many to solve with currently available technology. As proof of concept, we alternatively have solved simplified versions of multi-player Can't

Stop. The simplified games use dice with fewer than six sides, may have shorter columns than the official version, and may award a game to a player for completing fewer than 3 columns. Let (p, n, k, c) Can't Stop denote the p-player game played with n-sided dice and columns of length $k, k{+}2, \ldots, k{+}2(n{-}1), \ldots, k$ that is won when a player completes c columns.

We have implemented Algorithm 3 in Java and solved (p, n, k, c) Can't Stop for six combinations of small values of p, n, k, and c. Note that, in all cases, if $n = 2$ then $c = 1$ and if $n = 3$ then $c = 2$ because if we allow larger values of c then the game may end with no player winning the required number of columns. We used an initial estimate of $(\frac{1}{p}, \ldots, \frac{1}{p})$ for the position values of the anchors within a component. We assume that, when a player has a choice of two or more moves that would maximize his expected score but would have different effects on the other players, he makes the same choice each time; exactly which choice is made is determined arbitrarily by the internal ordering of the moves.

Table 1 shows, for the six examined versions of the game, the size of the game graph, the time it took the algorithm to run, and the probability that the each player wins, assuming that each player plays optimally. The listed totals for components and positions within those components do not include the components that were not examined because of equivalence to other components (for (3,3,2,2) Can't Stop there were 4,539,783 such components).

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

(p, n, k, c)	Components	Positions	Time	P(win)			
				$P1$	$P2$	$P3$	$P4$
$(3, 2, 1, 1)$	13	207	0.375s	1.000	0.000	0.000	
$(3, 2, 2, 1)$	340	7,410	1.72s	0.804	0.163	0.0332	
$(3, 2, 3, 1)$	6,643	176,064	14.7s	0.717	0.217	0.0657	
$(3, 3, 1, 2)$	74,302	7,580,604	34m45s	0.694	0.230	0.0760	
$(3, 3, 2, 2)$	3,782,833	687,700,305	2d10h	0.592	0.277	0.130	
$(4, 3, 1, 1)$	48,279	2,168,760	19m30s	0.920	0.0737	0.00591	0.000474

In the five most simplified versions listed in Table 1, the probability of winning the game in a single turn is so high that the optimal strategy never chooses to end a turn early. Colluding players also have the same strategy: in order to prevent one player from winning it is best in these small versions to try to win straightforwardly on the current turn. Because no player ever ends a turn with partial progress in a column, there is never a question of whether or not to avoid an otherwise desirable column to benefit an ally. For (3,3,2,2) Can't Stop there are a few circumstances in which players should end their turns early; this allows a modest gain from collusion: players two and three can reduce player one's chance of winning by about 0.03%.

6 Our Findings

We used a $2n$-partite graph to abstract an n-player probabilistic game. Given a position u, its position value is a vector $f(u) = (f_1(u), \ldots, f_n(u)) \in [0,1]^n$, with $f_i(u)$ indicating the winning rate of the ith player for $i = 1, \ldots, n$. We investigated the game of multi-player Can't Stop. To obtain the optimal solution, we generalized an approximation algorithm from [8, 9, 5] by incorporating the n-dimensional Newton's method with retrograde analysis. The technique was then used to solve simplified versions of three- and four-player Can't Stop. The official versions of three- and four-player Can't Stop have too many components 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. D.P. Bertsekas. *Dynamic Programming and Optimal Control*, volume I. Athena Scientific, 3rd edition, 2005.
2. D.P. Bertsekas. *Dynamic Programming and Optimal Control*, volume II. Athena Scientific, 3rd edition, 2007.
3. K. Binmore. *Playing for Real: A Text on Game Theory.* Oxford University Press, USA, 2007.
4. J.E. Dennis and R.B. Schnabel. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations.* SIAM, Philadelphia, PA, USA, 1996.
5. H.-r. Fang, J. Glenn, and C.P. Kruskal. Retrograde approximation algorithms for jeopardy stochastic games. *ICGA Journal*, 31, 2008.
6. J. Glenn. An optimal strategy for Yahtzee. Technical Report CS-TR-0002, Loyola College in Maryland, 4501 N. Charles St, Baltimore MD 21210, USA, May 2006.
7. J. Glenn. Computer strategies for solitaire yahtzee. In *IEEE Symposium on Computational Intelligence and Games (CIG 2007)*, pages 132–139. 2007.
8. J. Glenn, H.-r. Fang, and C.P. Kruskal. A retrograde approximate algorithm for one-player Can't Stop. In H.J. van den Herik, P. Ciancarini, and H.H.L.M. Donkers, editors, *Lecture Notes in Computer Science, Vol. 4630*, pages 148–159, New York, NY, 2007. Springer-Verlag. CG'06 Proceedings.
9. J. Glenn, H.-r. Fang, and C.P. Kruskal. A retrograde approximate algorithm for two-player Can't Stop. In *CGW 2007 Workshop*, pages 145–156, Amsterdam, The Netherlands, 2007.
10. T. Neller and C. Presser. Optimal play of the dice game Pig. *The UMAP Journal*, 25(1):25–47, 2004.
11. J.W. Romein and H.E. Bal. Solving the game of Awari using parallel retrograde analysis. *IEEE Computer*, 36(10):26–33, October 2003.
12. 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.
13. H.J. van den Herik, J.W.H.M. Uiterwijk, and J. van Rijswijck. Games solved: Now and in the future. *Artificial Intelligence*, 134:277–311, 2002.
14. P. Woodward. Yahtzee: The solution. *Chance*, 16(1):18–22, 2003.