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

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

[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
University of Maryland
A.V. Williams Building, College Park, Maryland 20742, USA
{hrfang,kruskal}@cs.umd.edu

Abstract. A one-player, finite, probabilistic game with perfect information can be presented as a bipartite graph. For one-player Can't Stop, the graph is cyclic and the challenge is to determine the game-theoretical values of the positions in the cycles. In this article we prove the existence and uniqueness of the solution to one-player Can't Stop, and give an efficient approximation algorithm to solve it by incorporating Newton's method with retrograde analysis. We give results of applying this method to small versions of one-player Can't Stop.

1 Introduction

Retrograde analysis has been well developed and successfully applied to deterministic, finite, two-player zero-sum games with perfect information, such as Awari [2], checkers [3], and chess [4]. For some probabilistic games, such as Yahtzee and Can't Stop[1], retrograde algorithms are less practical due to the high complexity. Therefore, we studied the simplified one-player game instead.

A one-player probabilistic game can be represented as a bipartite graph, in which one set of nodes corresponds to deterministic events while the other corresponds to random events. For one-player Yahtzee, the graph representation is acyclic, which simplifies algorithm design and allows the game to be solved easily [1] [5]. In some games, the graph representation is cyclic, which causes difficulty in designing a bottom-up retrograde algorithm. We are particularly interested in one-player Can't Stop, developed an approximation algorithm to solve this game by incorporating Newton's method with retrograde analysis, and give results of applying the method to small versions of the game.

[1] Can't Stop was designed by Sid Sackson and marketed by Parker Brothers. It is currently out of print but will be republished by Face 2 Face Games in 2006. The rules can be found at http://en.wikipedia.org/wiki/Can't_Stop and http://www.boardgamegeek.com/game/41. Also see Appendix for a short description.

The organization of this paper is as follows. Section 2 formulates the problem. In Section 3 we prove that one-player Can't Stop has a unique solution, and give an efficient 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. A short description of Can't Stop game rules is given in Appendix.

2 Problem Formulation

A one-player, finite and probabilistic game with perfect information can be represented as a directed, bipartite *game graph* $G = (U, V, E)$, where U and V are two disjoint sets of vertices and E is the set of edges. (An edge (x, y) must have either $x \in U$ and $y \in V$ or $x \in V$ and $y \in U$.) The graph may be cyclic. A *position* is a vertex $w \in U \cup V$.

In Yahtzee a turn consists of a dice roll followed by a move. In Can't Stop a turn consists of a sequence of partial turns, each of which is a dice roll followed by a move. A *roll position* is a vertex $u \in U$. A *move position* is a vertex $v \in V$.

For each non-terminal roll position u, a *dice roll* is a random event. The weight $0 < p((u, v)) \leq 1$ indicates the probability that the game in roll position u will change into move position v. So,

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

A *move* (v, u) (from a move position to a roll position) is a deterministic choice.

A *partial turn* (u_1, u_2) *(from roll position u_1 to roll position u_2)* consists of a dice roll followed by a move. It is represented by a pair of edges $((u_1, v), (v, u_2))$ in G. A *turn* is a sequence of partial turns $(u_0, u_1), (u_1, u_2), \ldots, (u_{k-1}, u_k)$. As noted above, in Yahtzee, a turn consists of exactly one partial turn, and in Can't Stop a turn may consist of many partial turns.

We associate each vertex with a number, as the expected cost (or penalty) of playing the optimal strategy starting from that vertex. The information is stored in a *cost database*, which is presented as a function $f : (U \cup V) \to R$.

The goal of one-player Can't Stop is to play so as to minimize the expected number of turns to finish the game. Therefore, $f(u)$ is the expected number of remaining turns to finish the game starting at roll position u, in optimal play.

The cost function f satisfies that for all non-terminal roll positions $u \in U$,

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

where $g(u)$ is the step cost (or step penalty) at u.

In the game of one-player Can't Stop, $g(u)$ indicates whether it is the first partial turn for a turn. More precisely,

$$g(u) = \begin{cases} 1, & \text{if } u \text{ is the starting position for a turn,} \\ 0, & \text{otherwise.} \end{cases}$$

For the optimal playing strategy, we minimize the cost (or penalty)[2]. Therefore, for all non-terminal move positions $v \in V$,

$$f(v) = \min_{\forall u \text{ with } (v,u) \in E} f(u). \tag{2}$$

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

A terminal vertex indicates the end of a game. We assume all terminal vertices, denoted by z, are roll positions (in U) with $f(z) = g(z)$. For one-player Can't Stop, a terminal vertex z is reached when the player completes three columns, and therefore no additional rolling of dice is required (i.e., $f(z) = g(z) = 0$).

A cost 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 stand for finite, one-player, probabilistic games with perfect information.

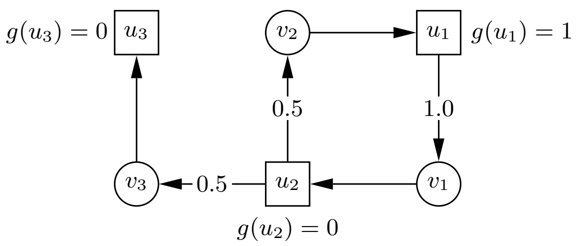

Fig. 1. An example of game graph $G = (U, V, E)$.

We illustrate an example in Figure 1, where $g(u_1) = 1$, $g(u_2) = g(u_3) = 0$, $p((u_1, v_1)) = 1$, and $p((u_2, v_2)) = p((u_2, v_3)) = 0.5$. This example simulates the last stage of a game of one-player Can't Stop. A turn begins at position u_1. At position u_2, the first player has 50% probability to complete three columns, and the other 50% probability to fall back to u_1. By (1) and (2),

$$\begin{aligned} f(v_1) &= f(u_2) = \frac{1}{2} f(v_2) + \frac{1}{2} f(v_3), \\ f(v_2) &= f(u_1) = f(v_1) + 1, \\ f(v_3) &= f(u_3) = 0. \end{aligned}$$

The unique solution is $f(u_1) = f(v_2) = 2$ and $f(u_2) = f(v_1) = 1$.

The problem of solving a one-player probabilistic game is formulated as follows. Suppose we are given a game graph $G = (U, V, E)$ of a one-player, finite, probabilistic game with perfect information, and its step cost function $g : U \to R$. First, we investigate the existence and uniqueness of the solution

[2] If the goal of a game is to maximize some value, we can multiply it by -1 and minimize.

(i.e., the cost database $f : (U \cup V) \rightarrow R$ that satisfies both conditions (1) and (2)). Second, we design an efficient algorithm to construct the cost database, assuming a solution exists.

3 Retrograde Analysis for One-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 cost database of an acyclic game graph. In Subsection 3.2, we prove that one-player Can't Stop has a unique solution. In Subsection 3.3, we give an approximation algorithm to construct the cost database for a game graph with cycles.

3.1 Game Graph is Acyclic

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

Consider Algorithm 1. Assuming all terminal vertices are in 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 call a vertex *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 acyclic property ensures that all vertices are determined at the end of propagation phase. Therefore, a final phase is not required. The optimal playing strategy is clear: given $v \in V$, always make the move (v, u) with the minimum $f(u)$.

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.

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

Proof. In an acyclic graph, the level 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. □

Algorithm 1 Construct cost database f for acyclic game graph $G = (U, V, E)$.

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

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 prove that the game graph of one-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. *A cyclic game graph $G = (U, V, E)$ has a solution if,*

1. *For all $u \in U$, $g(u) \geq 0$.*
2. *For each non-terminal vertex u, there is a path from u to a terminal vertex.*
3. *There exists some $w \in U$ such that the graph is acyclic after removing the outgoing edges of w.*

[3] See, for example, `http://mathworld.wolfram.com/FixedPointTheorem.html`.

In addition, if the vertex w in condition 3 satisfies $g(w) > 0$, then the solution is unique with all position values non-negative.

Proof. Let $\hat{G} = (U, V, \hat{E})$ be the graph obtained by removing all of the outgoing edges from w in G (i.e., $\hat{E} = \{(u, v) : (u \in U - \{w\}) \wedge (v \in U \cup V) \wedge ((u, v) \in E)\}$). By condition 3, $\hat{G} = (U, V, \hat{E})$ is acyclic. All the terminal vertices other than w in $\hat{G}$ are also terminal in G. Let x be the estimated position value of w. We can construct a database for $\hat{G}$ by Algorithm 1. However, we propagate in terms of x (i.e., treat x as a variable during the propagation), though we know the value of x. For example, assuming $x = 6$, we write $\min\{\frac{1}{2}x, \frac{1}{3}x + 2\} = \frac{1}{2}x$ instead of 3. We use $\hat{f}(x, y)$ to denote the position value of $y \in U \cup V$ of $\hat{G}$ in terms of x. At the end of Algorithm 1, we compute $\hat{f}(x, w)$ with edges in $E - \hat{E}$ in terms of x by (1). The values of $\hat{f}(x, y)$ for all $y \in U \cup V$ constitute a solution to G, if and only if $\hat{f}(x, w)$ equals x in value. The main theme of this proof is to discuss the existence and uniqueness of x satisfying $\hat{f}(x, w) = x$.

Iteratively applying (1) and (2), all $\hat{f}(x, y)$ for $y \in U \cup V$ are in the form $ax + b$, where $0 \leq a \leq 1$. We are particularly concerned with $\hat{f}(x, w)$. Let $\hat{f}(x, w) = a(x)x + b(x)$, where $a(x)$ and $b(x)$ are real functions of x. By (1) and (2), it is not hard to see that $a(x)$ is non-increasing, $b(x)$ is non-decreasing, and both $a(x)$ and $b(x)$ are piecewise constant. Hence $\hat{f}(x, w)$ is piecewise linear, continuous and non-decreasing in terms of x. By condition 1, $\hat{f}(0, w) = b(0) \geq g(w) \geq 0$. By condition 2, $a(x) < 1$ for x large enough. Since $a(x)$ is non-increasing and $a(x) < 1$ for x large enough, $f(x) < x$ for x large enough. By Theorem 1, there exists $x \geq 0$ such that $\hat{f}(x, w) = x$.

By condition 1, $\hat{f}(0, w) \geq g(w)$. Assuming $g(w) > 0$, then $\hat{f}(w, 0) > 0$. Moreover, $\hat{f}(x, w) = a(x)x + b(x)$ is piecewise linear and continuous with $0 \leq a(x) \leq 1$ for $x \in R$. Therefore, $x \leq 0$ implies $\hat{f}(w, x) > x$. Let $x_0 > 0$ be the smallest solution to $\hat{f}(w, x) = x$. Since $\hat{f}(w, 0) > 0$ and $a(x)$ is non-increasing, $a(x_0) < 1$ and therefore $\hat{f}(w, x) < x$ for $x > x_0$. We conclude that the additional condition $g(w) > 0$ guarantees the solution $x_0 > 0$ to $\hat{f}(w, x) = x$ is unique. Hence the game graph G has a unique solution with all position values non-negative. □

Consider the strongly connected components of the game graph of one-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. The roll position with no marker is the *anchor* of the component. When left without a legal move, the game goes back to the anchor, 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 the player has won three columns. Each terminal component has only one vertex with position value 0.

Theorem 2. *The game graph of one-player Can't Stop has a unique solution.*

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 non-negative and uniquely determined, we consider the subgraph induced by the component and the anchors in its supporting components. This subgraph satisfies all the 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 non-negative. By induction, the solution to the game graph of one-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 $G = (U, V, E)$, we prune some edges so the resulting $\hat{G} = (U, V, \hat{E})$ 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 a cost 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 $G = (U, V, E)$.

```
Ensure: If $\hat{f}$ converges, it converges to a solution to $G = (U, V, E)$.
  Obtain an acyclic graph $\hat{G} = (U, V, \hat{E})$, 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 $G$.
  $S_1 \leftarrow \{$terminal positions of $\hat{G}$ in $U\}$
  $S_2 \leftarrow \{$terminal positions of $\hat{G}$ in $V\}$
  repeat                                                                               ▷ Approximation Phase
      for all $u \in S_1$ do
          for all $(v, u) \in E$ do
              $\hat{f}(v) \leftarrow \min_{\forall w \text{ with } (v,w) \in E} \hat{f}(w)$      ▷ (*)
              $S_2 \leftarrow S_2 \cup \{v\}$
          end for
      end for
      $S_1 \leftarrow \emptyset$
      for all $v \in S_2$ do
          for all $(u, v) \in E$ 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
      end for
      $S_2 \leftarrow \emptyset$
  until $\hat{f}$ converges.
```

An example is illustrated by solving $G = (U, V, E)$ in Figure 1. We remove (u_1, v_1) to obtain the acyclic graph $\hat{G}$. The newly terminal vertex is u_1. Let $\hat{f}(u_1) = 1$, which is a reasonable initial guess since $f(u_1) \geq g(u_1) = 1$ in G. The solution for $\hat{G}$ is $\hat{f}(u_1) = \hat{f}(v_2) = 1$ and $\hat{f}(u_2) = \hat{f}(v_1) = \frac{1}{2}$. The update is repeated with $\hat{f}(u_1) = \frac{3}{2}, \frac{7}{4}, \ldots, \frac{2^{n+1}-1}{2^n}, \ldots$, which converges to 2. 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 $\frac{e_{n+1}}{e_n} = \frac{1}{2}$. In other words, it converges linearly.

Consider Algorithm 2. For one-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 cost changes of the children is sufficient.

If the conditions in Lemma 2 are satisfied (e.g., a strongly connected component of one-player Can't Stop), $\hat{G}$ can be obtained by pruning the outgoing edges of the anchor w. In this case, Algorithm 2 corresponds to the steepest descent method without line search in numerical optimization, so linear convergence is expected.

The proof of Lemma 2 reveals that if we solve $f(x, w) = x$ using 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 in Lemma 2, and let x be the initial estimate of the position value of u_1. Then $\hat{f}(x, v_2) = x$, $\hat{f}(x, v_1) = \hat{f}(x, u_2) = \frac{1}{2}x$, and $\hat{f}(x, u_1) = \frac{1}{2}x + 1$. Solving $\frac{1}{2}x + 1 = x$, we obtain $x = 2$, which is the exact position value of u_2. 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.

Consider Algorithm 3. In the estimation phase, the better the initial estimate of position value of w (denoted by x), the fewer steps are needed to reach the solution.

4 Indexing Scheme

In practice, a game graph $G = (U, V, E)$ can be too big to fit in physical memory. For one-player Can't Stop, we partition the graph into strongly connected components. Before constructing a cost database for a component, we have all its supporting databases constructed. The construction is in bottom-up order, until the game is solved.

[4] See, for example, `http://mathworld.wolfram.com/NewtonsMethod.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.

Algorithm 3 An efficient algorithm to solve a game graph with one anchor.

```
Require: $G = (U, V, E)$ satisfies the conditions in Lemma 2.
Ensure: $\hat{f}$ converges to a solution to $G$ in the rate of Newton's method.
  Let $x$ denote the estimate for position value of $w$ in Lemma 2.  ▷ Estimation Phase
  Obtain the acyclic graph $\hat{G} = (U,V,\hat{E})$ by removing the outgoing edges of $w$.
  repeat                                                          ▷ Approximation Phase
    Solve $\hat{G}$ (in terms of $x$) with the current estimate $x$ for $w$ by Algorithm 1.
    Compute $\hat{f}(x, w)$ with $E - \hat{E}$ by (1) in terms of $x$. Denote the result by $ax + b$.
    $x \leftarrow \frac{b}{1-a}$.              ▷ (The solution to $ax + b = x$ is $x = \frac{b}{1-a}$.)
  until $\hat{f}(x, w) = x$ in value.
```

4.1 Indexing Scheme for Can't Stop

Consider one-player Can't Stop. Let x_i denote the number of steps from the square to the top at column 'i'. Each strongly connected component of the game graph consists of all the positions with some particular $(x_2, x_3, \ldots, x_{12})$, where $0 \leq x_i \leq 2i - 1$ for $i = 2, 3, \ldots, 7$ and $0 \leq x_i \leq 27 - 2i$ for $i = 7, 8, \ldots, 12$ and at most three of the x_i are zero. $(x'_2, x'_3, \ldots, x'_{12})$ is a supporting component of $(x_2, x_3, \ldots, x_{12})$ if and only if $x'_i \leq x_i$ for $i = 2, 3, \ldots, 12$ and $(x'_2, x'_3, \ldots, x'_{12}) \neq (x_2, x_3, \ldots, x_{12})$.

A terminal component has three zero squares, and contains only one position in the game graph G (win three columns, the end of a game). Each position in a non-terminal component $(x_2, x_3, \ldots, x_{12})$ is $(y_2, y_3, \ldots, y_{12})$ where each $y_i \leq x_i$ and for at most three i, $y_i < x_i$. All positions are indexed using

$$\sum_{c=2}^{12} y_c \prod_{d=2}^{c-1} l(d)$$

as a hash value, where $l(d)$ denotes the length of column d. When the cost database cannot fit in main memory, it is possible to reorder the columns (e.g., consider a position to be $(x_2, x_{12}, x_3, x_{11}, \ldots)$ if that provides for better access patterns). The cost databases for the anchor positions and the non-anchor positions are maintained separately, since the position values of the non-anchor positions in a component are used only when computing the position value for the anchor in that component.

In practice, we discard the cost database for the non-anchor positions and reconstruct them using the cost database for the anchor positions as necessary (as when simulating perfect play). If storage is abundant and speed is important, it would also be possible to use a file named $x_2x_3 \ldots x_{12}.ijk$ to store all the position values, where $i < j < k$ are the columns where $x_i \neq y_i$, $x_j \neq y_j$, and $x_k \neq y_k$. The offset of a position would be $y_i + y_j x_i + y_k x_i x_j$ in the file. The naming and indexing convention is the same for positions with two markers or fewer. The cost database for component $(x_2, x_3, \ldots, x_{12})$ consists of all files $x_2x_3 \ldots x_{12}*$. The largest component is $(3, 5, 7, 9, 11, 13, 11, 9, 7, 5, 3)$, which contains the position of the beginning of the game.

4.2 Algorithms

When the game graph is split, we can construct the cost databases component-by-component in the bottom-up order. Algorithm 2 is applied to the subgraph consisting of the component, its outgoing edges, and the positions that the outgoing edges connect to. For each supporting component, there is only one position the parent component connects to, the anchor (i.e., the position with no markers). In Algorithm 2, we may propagate the information from the supporting databases to the component, so the supporting databases are not required in the propagation phase.

5 Experiments

As proof of concept, we have solved simple versions of one-player Can't Stop. These simpler versions use 3-, 4-, and 5-sided dice instead of 6-sided dice and may have shorter columns than the official version. Let (n, k) Can't Stop denote the one-player game played with four n-sided dice with the shortest column k spaces long. Columns 2 and $2n$ are the shortest columns and column $n + 1$ is the longest. Adjacent columns always differ in length by 2 spaces. The official version is then $(6, 3)$ Can't Stop.

For $n = 2, 3, 4$ and $k = 1, 2, 3$ (and also $n = 5, k = 1$) we have implemented Algorithm 3 in Java and solved (n, k) Can't Stop. We used an initial estimate of 1.0 for the position value of each vertex. Table 1 shows, for each version of the game, the size of the game graph, the time it took the computer to solve the game, and the average number of turns needed to win the game when using the optimal strategy. The size of the game graph is given as the number of anchor vertices (i.e., vertices representing the beginning of a turn with no markers placed), and the total number of vertices in all of the anchor vertices' strongly connected components (which includes vertices representing the middle of a turn when the markers have been placed on the board). Symmetry allows us to ignore about half of the anchor vertices in our implementation since the position represented by $(x_2, x_3, \dots, x_{12})$ is equivalent to $(x_{12}, x_{11}, \dots, x_2)$.

Note that for fixed n, the time to solve the game is roughly proportional to the number of vertices. When n increases there is also an additional cost due to the increased number of outgoing edges from each vertex in U. For $n = 3$ there are 15 neighbors of each vertex (representing the 15 different outcomes of rolling four 3-sided dice); for $n = 4$ there are 35 neighbors.

The average position value also affects the running time. For larger values of k or n, the average position value is higher; higher position values will require more iterations in Algorithm 3 to converge. Table 2 shows the number of iterations required for convergence when solving $(4, 3)$ Can't Stop.

6 Conclusion

We used a bipartite graph to abstract a one-player probabilistic game with the goal to maximize some expected game value or to minimize the expected cost.

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

(n,k)	Anchor vertices	Total vertices	Time	Optimal Turns
$(2,1)$	15	225	0.166s	1.298
$(2,2)$	44	1,936	0.405s	1.347
$(2,3)$	95	9,025	0.601s	1.400
$(3,1)$	308	64,372	1.70s	1.480
$(3,2)$	1,432	787,600	5.05s	1.722
$(3,3)$	4,378	4,934,006	23.3s	1.890
$(4,1)$	12,913	20,802,843	4m50s	2.187
$(4,2)$	83,456	289,091,584	58m50s	2.454
$(4,3)$	333,069	2,104,663,011	6h7m	2.700
$(5,1)$	921,174	7,105,015,062	2d20h	2.791

Table 2. For Can't Stop $(4,3)$, # of iterations required for ranges of position values.

Position Value	States	Mean Iter.	Position Value	States	Mean Iter.
1.0 - 1.1	50,044	3.31	1.9 - 2.0	6,326	8.27
1.1 - 1.2	21,147	3.41	2.0 - 2.1	8,096	8.32
1.2 - 1.3	8,842	3.73	2.1 - 2.2	8,797	8.61
1.3 - 1.4	13,535	4.32	2.2 - 2.3	7,598	8.90
1.4 - 1.5	9,617	5.00	2.3 - 2.4	5,210	9.18
1.5 - 1.6	5,829	5.88	2.4 - 2.5	2,574	9.43
1.6 - 1.7	3,524	6.70	2.5 - 2.6	684	9.61
1.7 - 1.8	3,157	7.51	2.6 - 2.7	75	9.76
1.8 - 1.9	4,321	8.10	Total	159,376	5.13

We investigated the game of one-player Can't Stop, and proved that its optimal solution exists and is unique. To obtain the optimal solution, we developed a new approximation algorithm that converges quadratically, by incorporating Newton's method with retrograde analysis.

We successfully constructed the databases of the simplified models with 3-sided and 4-sided dice. The optimal solution of one-player Can't Stop can be used as the approximate solution of two-player Can't Stop. Two-player Can't Stop can be presented as a four-partite graph. Given a position, a function f is defined as the probability that the first player wins the game. The goal is to build a database representing f. In addition to building the optimal databases of one-player Can't Stop, we plan to tackle two-player Can't Stop in the future.

7 Acknowledgements

The authors thank David Slater for pointing out that the algorithm can be improved by taking advantage of symmetry.

References

1. 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.
2. J.W. Romein and H.E. Bal. Solving the game of awari using parallel retrograde analysis. *IEEE Computer*, 36(10):26–33, October 2003.
3. 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.
4. K. Thompson. 6-piece endgames. *ICCA Journal*, 19(4):215–226, 1996.
5. P. Woodward. Yahtzee: The solution. *Chance*, 16(1):18–22, 2003.

Appendix: Can't Stop Rules

We summarize the game rules of Can't Stop, largely taken from Wikipedia[6]: The game equipment consists of four dice, a board, a set of eleven markers for each player, and three neutral markers. The board consists of eleven columns of spaces, one column for each of the numbers 2 through 12. The columns (respectively) have 3, 5, 7, 9, 11, 13, 11, 9, 7, 5 and 3 spaces each. The object of the game is to move your markers up the columns, and be the first player to complete three columns.

On a player's turn he[7] rolls all four dice. He then divides the four dice into two pairs, each of which has an associated total. (For example, if he rolled 1 - 3 - 3 - 4 he could make a 4 and a 7, or a 5 and a 6.) If the neutral markers are off of the board then they are brought on to the board on the columns that correspond to these totals. If the neutral markers are already on the board in one or both of these columns then they are advanced one space upward. If the neutral markers are on the board, but only in columns that cannot be made with any pair of the current four dice, then the turn is over and the player gains nothing.

After moving the markers the player chooses whether or not to roll again. If he stops, then he puts markers of his color in the locations of the current neutral markers. If on a later turn he restarts this column, he starts building from the place he previously claimed. If he does not stop then he must be able to advance at least one of the neutral markers on his next roll, or all progress on this turn is lost.

When a player reaches the top space of a column, that column is won, and no further play in that column is allowed. The first player to complete three columns wins the game.

[6] `http://en.wikipedia.org/wiki/Can't_Stop`

[7] We use 'he' when both 'she' and 'he' are possible.