Proceedings of the 2007 IEEE Symposium on Computational Intelligence and Games (CIG 2007)

Computer Strategies for Solitaire Yahtzee

James R. Glenn
Department of Computer Science
Loyola College in Maryland
jglenn@cs.loyola.edu

***Abstract*— Solitaire Yahtzee has been solved completely. However, the optimal strategy is not one a human could practically use, and for computer play it requires either a very large database or significant CPU time. We present some refinements to the techniques used to solve solitaire Yahtzee and give a method for analyzing other solitaire strategies and give some examples of this analysis for some non-optimal strategies, including some produced by evolutionary algorithms.**

Keywords: computer games, retrograde analysis, evolutionary computation, non-deterministic games

I. Introduction

Solitaire Yahtzee has been studied extensively: around the same time, several groups independently computed the optimal strategy for different versions of the game [1] [2] [3]. In all cases, the optimal strategy was considered to be the strategy that maximizes the expected score. Four years later, Woodward repeated the computations and published the results [4]. We present some non-optimal strategies, including some a human could easily follow, and a technique for analyzing them. We also present some refinements to the earlier work that significantly decrease the time required to compute the optimal strategy; these refinements are used in the analysis of non-optimal strategies as well.

II. Optimal Solitaire Yahtzee

Yahtzee is a game played with five six-sided dice and a scoresheet listing 13 categories. On each turn, the player rolls the dice, rerolls any subset of them zero, one, or two times (it does not have to be the same subset rerolled the second time), and then chooses a category in which to score the final roll. Each category has its own rules for scoring (see Table I), and each category may be used only once per game.

In addition to the rules in Table I, there are two possibilities for bonus points. One is a 100 point bonus for the second and subsequent Yahtzees, provided that the first was scored as 50 points in *Yahtzee*. The second is a 35 point bonus for achieving at least 63 points in the first six categories (corresponding to three of each matching die in each category). For this reason, it is useful to split the categories into the Upper Categories and the Lower Categories.

Definition 1: $\mathcal{C}$, $\mathcal{U}$, and $\mathcal{L}$ are sets of Yahtzee categories as defined below:

1) $\mathcal{C}$ is the set of all categories $\{1, 2, 3, 4, 5, 6, 3K, 4K, FH, SS, LS, C, Y\}$;
2) $\mathcal{U}$ is the set of upper categories $\{1, 2, 3, 4, 5, 6\}$; and
3) $\mathcal{L}$ is the set of lower categories $\mathcal{C} - \mathcal{U}$.

TABLE I
Scoring Rules

Category	Rule
Aces	one point for every die showing one pip
Deuces	two points for each two
Treys	three points for each three
Fours	four points for each four
Fives	five points for each five
Sixes	six points for each six
Three of a Kind	the total on all the dice provided that three show the same number; zero otherwise
Four of a Kind	the total on all the dice provided that four show the same number; zero otherwise
Full House	25 points if three dice show one number and two show another; zero otherwise
Small Straight	30 points for four dice showing consecutive numbers; zero otherwise
Large Straight	40 points for five dice showing consecutive numbers; zero otherwise
Chance	the total on all the dice
Yahtzee	50 points if all five dice show the same number; zero otherwise

During a game the player has many choices to make. For example, having finished a turn with [5 5 6 6 6], a player may choose to score the roll in (among other choices) *Sixes*, *Full House*, or *Three of a Kind*. Having rolled [1 1 1 3 3] at the beginning of a turn, the player has to choose whether to reroll no dice and take a *Full House*, or risk losing the *Full House* to try for *Yahtzee*. The optimal strategy specifies which choices will maximize the expected score.

A. Solitaire Yahtzee Position Graph

Games like Yahtzee that are characterized by random events followed by player reponses to those random events can be modelled as bipartite graphs in which one set of vertices represents states of the game in which a random event is about to happen, and the other set represents states in which the player has a choice to make [5]. The previous work indeed treats solitaire Yahtzee in this way. We quickly present the technique used by Glenn [1], Holderied [3], Verhoeff [2], and Woodward [4] to compute the optimal strategy. Some important differences in the work of the four will be noted in a subsequent section.

It is useful to refine the graph further by grouping vertices together according to where in a turn they occur in addition to whether the next move is a random event or a player choice. Solitaire Yahtzee can then be viewed as a 6-partite graph $G = (V_1, V_2, \ldots, V_6, E)$ where $E \subset (V_1 \times V_2) \cup (V_2 \times V_3) \times$

$\cdots \times (V_5 \times V_6) \times (V_6 \times V_1)$. Vertices in V_1 are positions at the start of turns, vertices in V_2 represent the outcome of the initial roll, vertices in V_3 represent the choices of which dice to keep, and so on. We need enough vertices so that all of the information relevant to the optimal strategy can be encoded in the vertices. The necessary information includes what categories have been used, but not what scores have been obtained in those categories. In other words, a player who has scored 30 in *Small Straight* and 40 in *Large Straight* should play exactly the same way as a player who has scored zero in those two categories. Both players play to maximize their *future* score without regard to what has been done in the past. There are two exceptions: the total in the upper categories is relevant because of the upper bonus, and whether the Yahtzee category has 0 or 50 is relevant because of the 100-point bonus for extra Yahtzees (the Yahtzee Joker rule governing when a Yahtzee can be used as *Full House* or one of the straights does not affect the number of positions).

Therefore, we view each $v \in V_1$ as a triple $(U, upper, extras) \in \mathcal{C} \times \{0, \ldots, 63\} \times \{0, 1\}$, where U is the set of categories that have been used at the corresponding position, $upper$ denotes the upper total (values greater than 63 are equivalent to 63 since then the bonus has already been earned), and $extras$ is a flag that indicates whether the player earns the 100-point bonus for extra Yahtzees. There are then $2^{13} \cdot 64 \cdot 2 \cdot \frac{3}{4} = 786,432$ total positions in V_1 (taking advantage of the fact that we cannot have $extras = 1$ if the Yahtzee category has not been used), of which at most 314,880 are needed once reachability issues and equivalence have been taken into consideration (for example, it is impossible that $upper > 5$ if *Aces* is the only upper category used, and any position $(U, upper, extras)$ in which it is impossible to obtain the bonus is equivalent to $(U, 0, extras)$).

Vertices $u \in V - V_1$ represent positions in the middle of a turn. For such u there is a unique articulation point $v \in V_1$ such that any path from the initial position to u goes through v. We call all of the positions with a common articulation point a *component* and denote a component by G_u where u is the position in the component that is in V_1 (that is, it is the position of the start of a turn). We refer to u as the *anchor* of the component. For convenience, we refer to non-anchors by listing the anchor and the state of the dice. All of the vertices in the graph are then n-tuples.

1) A vertex $u \in V_1$ is $(U, upper, extras)$ as described above.
2) A vertex $v \in V_2 \cup V_4 \cup V_6$ represents a position after the dice have been rolled and will be written (u, R) where u is the anchor such that $v \in G_u$ and R (the roll) is a 5-element multiset with elements from $\{1, 2, 3, 4, 5, 6\}$ representing the outcome of rolling five 6-sided dice (note there are 252 such outcomes).
3) A vertex $v \in V_3 \cup V_5$ represents a position after the player has selected which dice to keep and will be written (u, R) where u is again the anchor and R is now a k-element multiset with elements from $\{1, 2, 3, 4, 5, 6\}$ for some $0 \leq k \leq 5$ (note there are 462 possible values of R).

Some vertices within the same component may share the same label (for example, [1 2 4 4 5] may appear as the inital roll, or after the first or second reroll); in the subsequent sections we will distinguish between them by explicitly stating which of the six vertex sets we are working in.

We describe the edges within each component The edges within a component largely do not depend on the anchor of the component. The edges leaving a component are dependent on the anchor, and for them we must know the score that would be obtained by moving along that edge.

Definition 2: For any positions $u \in V_6$ and $v \in V_1$

1) $S(u, v)$ denotes the score earned by moving from u to v, including any bonuses; and
2) $S'(u, v)$ denotes the score earned by moving from u to v, not including any bonuses.

Within a single component G_u (where $u = (U, upper, extras)$), the edges are as follows:

1) there is an edge from u to each vertex v in V_2;
2) if $v_1 = (u, R_1) \in V_2$ (or V_4) and $v_2 = (u, R_2) \in V_3$ (respectively V_5) then (u, v) exists exactly when $R_2 \subseteq R_1$;
3) if $v_1 = (u, R_1) \in V_3$ (or V_5) and $v_2 = (u, R_2) \in V_4$ (respectively V_6) then (u, v) exists exactly when $R_1 \subseteq R_2$; and
4) if $v = (u, R) \in V_6$ and $u' = (U', upper', extras') \in V_1$, then $(u, v) \in E$ if and only if $U' - U = \{c\}$ for some category c, $upper' = \min(upper + S'(u, v), 63)$ if $c \in \mathcal{U}$ and $upper' = upper$ otherwise, and $extras' = 1$ if and only if either $extras = 1$ or both $c = Y$ and $S(u, v) > 0$.

Note that the structure of one component is the same as the structure of any other, except for the edges of the last type, which go to the anchors of other components according to what categories are unused in U and what bonuses are earned when moving along those edges. We will call one component G_u a *neighbor* of another component G_v if there is an edge from G_v to G_u. This term can also be applied to anchors: the neighbors of an anchor u are the anchors of the neighbors of the component G_u.

Each component has 1,681 positions and up to 20,880 edges. There are 529,313,280 positions total in the 314,880 reachable components. Because the graph is acyclic (since once a category is used the score there can never be erased), we can compute the corresponding position values using retrograde analysis: start by computing the position values of the terminal positions, and then work backwards in order of reverse topological sort to the initial position of the game.

Definition 3: $X(u)$ denotes the *position value* of u, which is the expected future score at u.

The terminal positions of the game are those positions $u = (U, upper, extras)$ where $U = \mathcal{C}$. $X(u) = 0$ for any terminal position u. For any non-terminal position $u \in V_1 \cup V_3 \cup V_5$,

$$X(u) = \sum_{(u,v) \in E} P_{roll}(u, v) X(v) \tag{1}$$

where $P_{roll}(u, v)$ is the probability that v is the next position given that the current position is u (that is, the probability of the roll that moves the game from u to v). For any position in $u \in V_2 \cup V_4$,

$$X(u) = \max_{(u,v)\in E} X(v). \quad (2)$$

Finally, for any $u = ((U, upper, extras), R) \in V_6$,

$$X(u) = \max_{(u,v)\in E} S(u, v) + X(v). \quad (3)$$

The maximum expected future score of the initial position is the maximum expected score of the game. The entire computation can be done in well under an hour on current commodity desktop computers. $X(s)$ is approximately 254.59 for the initial position s; someone following the optimal strategy would expect to score 254.59 points.

B. Refinements of the Position Graph

We now present two refinements that speed up the computation of the optimal solitaire strategy by reducing the number of edges examined per component by approximately a factor of two. Since each edge is traversed once in the computations of $X(u)$ using formulas 1, 2, and 3, and evaluating these formulas represents almost all of the work the algorithm must perform, the time to compute the optimal strategy is proportional to the number of edges examined in the position graph. Therefore, halving the number of edges reduces the running time by a factor of two. One of the refinements is particular to Yahtzee; the other could be generalized to similar games.

Woodward did not make use of the fact that, in each component, it does not matter how one arrives at the second roll [4]: he treats the positions after the sequences of moves (Roll [1 2 3 3 5], Keep [3 3], Roll [3 3 3 3 6]) and (Roll [3 3 3 5 6], Keep [3 3 3], Roll [3 3 3 3 6]) as different positions, even though the optimal strategy does not depend on how the roll [3 3 3 3 6] was obtained. As a result, it took him "many computing days" to compute the optimal strategy. The others working on the problem used the structure of the components given in section II-A. Even this, however, is not ideal.

Consider the components as defined in Section II-A. The positions representing keeping [3 3] and [3 3 3] after the first roll both have edges to [3 3 3 3 6]. In fact, *any* position reachable from [3 3 3] is reachable from [3 3]. In general, if $v_1 = (u, R_1), v_2 = (u, R_2) \in V_3$ (or V_5) and $R_1 \subseteq R_2$ then whenever $(v_2, w) \in E$ then also $(v_1, w) \in E$. We wish to capture this fact in the position graph in order to reduce the number of edges.

Instead of viewing the next move from [3 3] as rolling the three dice that were not kept all at once, we view the next *three* moves as rolling *one* of the dice in turn. Then keeping [3 3] and rolling a 3 as the next move results in exactly the same position as keeping [3 3 3] in the first place. We have thus redefined the outgoing edges from positions in $V_3 \cup V_5$ as follows: if $v_1 = (u, R_1) \in V_3$ (or V_5) and $v_2 = (u, R_2) \in V_3$ (respectively V_5) then $(v_1, v_2) \in E$ if and only if $R_1 \subseteq R_2$. All edges $(v_1, v_2) \in (V_3 \times V_4) \cup (V_5 \times V_6)$ are removed, except for those where $v_1 = (u, R)$ and $\mid R \mid = 5$ (that is, only positions where there are no more dice to roll have edges to the next set of vertices).

Each of the positions representing the choice of keeping 4 or fewer dice now have only 6 outgoing edges; before they had 252, 126, 56, 21, or 6 depending on how many dice were kept. The positions where 5 dice have been kept have 1 outgoing edge each. There are then only 15,228 edges in each component.

The first refinement took advantage of a situation particular to Yahtzee where a move could be broken down into smaller moves. The second refinement has the potential to be used in a wider variety of circumstances.

Suppose there is a set of positions $C \subset V_3$ (or V_5) such that for every vertex $v_1 \in V_2$ (respectively V_4), there is a vertex $v_2 \in C$ such that $(v_1, v_2) \in E$ (so C in some sense covers V_2). Let $k = \min_{v\in C} X(v)$. Suppose further that $w \in V_3$ (V_5) is such that $X(w) < k$. Then w need not be considered as a destination vertex in Equation 2. If there is a collection of covers $C_1, \ldots, C_n$ then we let

$$k = \max_{1\le i\le n} \min_{v\in C_i} X(v) \quad (4)$$

and the same cutoff condition applies. In terms of Yahtzee, if we can find a set of keep positions that we can choose no matter what we roll, then we needn't consider any of the incoming edges to keep positions that are worse than the worst of that set. For example, we can always choose to reroll all five dice. If another choice of dice to keep is worse than keeping none of the dice, we do not have to consider it. We modify the standard retrograde approach to computing the position values within each component.

Algorithm 1 Computing $X(v)$ within a component G_u

Require: G_u is a component in a Yahtzee position graph $(V_1, \ldots V_6, E)$, $X(v)$ is known for all anchors v neighboring G_u, $C_1, \ldots, C_n$ is a collection of covers.

Ensure: Program terminates with Equations (1) satisfied for u.

$\forall v \in G_u$, $X(v) \leftarrow 0$ ▷ Initialization
$\forall v \in G_u \cap V_6$, compute $X(v)$ according to Equation 3
$\forall v \in G_u \cap V_5$, in reverse order of topological sort, compute $X(v)$ according to Equation 1
Compute k according to Equation 4
Run Algorithm 2 on $G_u \cap V_5$ using cutoff k.
$\forall v \in G_u \cap V_3$, in reverse order of topological sort, compute $X(v)$ according to Equation 1
Compute k according to Equation 4
Run Algorithm 2 on $G_u \cap V_3$ using cutoff k.
$\forall v \in G_u \cap V_1$, compute $X(v)$ according to Equation 1.

In order to keep the cutoff high, it is desirable to choose covers that contain good choices of which dice to keep. The covers should also be small, since all edges coming into a cover must be examined. We use Algorithm 1 with the set of covers given in Table II, and can eliminate an average of

Algorithm 2 Computing $X(v)$ for $G_u \cap (V_2 \cup V_4)$.

Require: G_u is a component in a Yahtzee position graph $(V_1, \ldots V_6, E)$, $L = G_u \cap V_3$ or $L = G_u \cap V_5$, $X(v)$ is known for all $v \in L$, $C_1, \ldots, C_n$ is a collection of covers, and k is a cutoff as computed by Equation 4.

Ensure: Program terminates with Equations (2) satisfied for all positions w that L neighbors.

```
for all v ∈ L do
    if X(v) > k or v ∈ C_1 ∪ ··· ∪ C_n then
        for all (w, v) ∈ E do
            X(w) ← max(X(w), X(v))
        end for
    end if
end for
```

3,762 edges per component, which is approximately 43% of the edges coming into $V_3 \cup V_5$. After both refinements, we examine, on average, 11,466 edges per component, 55% of the original 20,880.

TABLE II
KEEPS THAT COVER ALL ROLLS

C_1	[]		
C_2	[1 1]	[2 2]	[3 3]
	[4 4]	[5]	[6]
C_3	[1 1]	[2 2]	[3 3]
	[4 4]	[5 5]	[6 6]
	[1 2 3]	[4 5 6]	

C. Statistics for the Optimal Strategy

It seems unlikely that we can extract enough pithy information from the final database of position values so that a human could follow the optimal strategy without having access to the database. However, opportunities exist for using the optimal strategy to coach human players. One such coaching method beyond the obvious (simply show the human the optimal move at each position and let the human try to extract patterns) would be to examine the player's average score in each category and compare that average to that obtained when following the optimal strategy. This way the human player will know which categories he is overvaluing (those where his average exceeds the optimal strategy's) and which he is undervaluing. This style of coaching requires us to know the relevant statistics for the optimal strategy. The relevant values can be estimated by simulation, but because the variances are so high (see Table III) it would require many runs to estimate them with any precision. We now present a way to obtain these statistics exactly for the optimal strategy. These statistics form the basis of some of the non-optimal strategies presented in Section III.

The optimal strategy, or any other solitaire Yahtzee strategy, can be viewed as a function $o : V_2 \cup V_4 \cup V_6 \to V_3 \cup V_5 \cup V_1$ that determines which dice to keep after the first roll or second rolls, and what category to use at the end of a turn. We must require that if $o(u) = v$ then $(u, v) \in E$ (that is, the strategy must always choose a move that is legal).

TABLE III
CATEGORY STATISTICS FOR THE OPTIMAL STRATEGY

Category	$\bar{s}(c)$	$\sigma^2(c)$
Aces	1.8813	1.4786
Deuces	5.2825	3.9916
Treys	8.5693	7.3641
Fours	12.1583	10.8039
Fives	15.6874	14.8300
Sixes	19.1889	21.5585
Three of a Kind	21.6614	31.5904
Four of a Kind	13.0977	122.6289
Full House	22.5918	54.4056
Small Straight	29.4612	15.8734
Large Straight	32.7113	238.4223
Chance	22.0091	6.4477
Yahtzee	16.8683	558.8751
Upper Bonus	23.8413	266.0375
Yahtzee Bonus	9.5801	

Denote by $P_{visit}(u)$ and $P_{visit}(u, v)$ the probability that an anchor is visited and an edge is traversed (respectively) when following the optimal strategy. The values of P can be computed by forward induction: $P_{visit}(s) = 1$ for the initial position s, and for $(u, v) \in (V_1 \times V_2) \cup (V_3 \times V_4) \cup (V_5 \times V_6)$,

$$P_{visit}(u, v) = P_{visit}(u) P_{roll}(u, v). \tag{5}$$

For any other edge (u, v),

$$P_{visit}(u, v) = \begin{cases} P_{visit}(u), & \text{if } o(u) = v, \\ 0, & \text{otherwise.} \end{cases} \tag{6}$$

For any anchor u other than the initial position,

$$P_{visit}(u) = \sum_{(v,u) \in E} P_{visit}(v, u). \tag{7}$$

What we are most interested in are the values of $P_{visit}(u, v)$ for $u \in V_6$; from these values we can obtain the probability of scoring n points in category c, which we will denote $P_{score}(c, n)$: let $c \in \mathcal{C}$ be a category and denote the set of edges in $V_6 \times V_1$ that use category c by $E(c)$ (that is, $E(c)$ is the set of edges that can be written as $((U, upper, extras, R), (U \cup \{c\}, upper', extras')))$. Then

$$P_{score}(c, n) = \sum_{\substack{(u,v) \in E(c) \\ S'(u,v) = n}} P_{visit}(u, v) \tag{8}$$

All of these quantities can be computed in one pass through the position graph using Algorithm 3. The algorithm will work for any other deterministic solitaire Yahtzee strategy by replacing o with the appropriate function. We can easily modify the algorithm so it also computes the probability of earning the upper bonus (by treating it as a separate category) and the expected number of Yahtzee bonuses (by summing $P_{visit}(u, v)$ for all moves (u, v) that earn a Yahtzee bonus).

Once $P_{score}(c, n)$ is computed for all c and n it is trivial to compute the mean $\bar{s}(c)$ and variance $\sigma^2(c)$ for each category

Algorithm 3 Computing $P_{score}(c, n)$

Require: $G = (V_1, \ldots V_6, E)$ is a Yahtzee position graph, f is a Yahtzee strategy.

Ensure: Program terminates with Equations (7) and (8) satisfied.

$\forall c \in \mathcal{C}, 0 \leq n \leq 50$, $P_{score}(c, n) \leftarrow 0$ ▷ Initialization
$\forall u \in V_1$, $P_{visit}(u) \leftarrow 0$.
$P_{visit}(s) \leftarrow 1$ for the initial position s.
for all $u \in V_1$ in order of topological sort **do**
  $\forall v \in G_u - \{u\}, P_{visit}(u) \leftarrow 0$
  for all $v \in G_u - \{u\}$ in order of topological sort **do**
    if $v \in V_1 \cup V_3 \cup V_5$ **then** ▷ v is rolling the dice
      for all $(v, w) \in E$ **do**
        $p \leftarrow P_{roll}(v, w) \cdot P_{visit}(v)$
        $P_{visit}(w) \leftarrow P_{visit}(w) + p$
      end for
    else if $v \in V_2 \cup V_4$ **then** ▷ choosing dice to keep
      $P_{visit}(f(v)) \leftarrow P_{visit}(f(v)) + P_{visit}(v)$
    else ▷ v is choosing a category
      write $v = (U, upper, extras, R)$
      write $f(v) = (U', upper', extras')$
      $c \leftarrow U' - U$ ▷ the strategy chooses category c
      $n \leftarrow S'(v, f(v))$
      $P_{score}(c, n) \leftarrow P_{score}(c, n) + P_{visit}(v)$
    end if
  end for
end for

c. Similar computations can be done for the statistics of the upper bonus. For the Yahtzee bonus we can compute only the mean; we cannot easily compute the probability of earning n points for the Yahtzee bonus because those points will have been earned while traversing several different edges. The values of $\bar{s}(c)$ and $\sigma^2(c)$ are given in Table III and agree with those computed by numerical approximation by Verhoeff [2].

D. Score Distribution for the Optimal Strategy

We can also compute, for each possible score n, the probability $P_{total}(n)$ that the optimal strategy scores n points:

$$P_{total}(n) = \sum_{\substack{u \in T \\ total(u) = n}} P_{visit}(u) \tag{9}$$

where T is the set of terminal positions and $total(u)$ denotes the total score obtained at position u. However, the positions as given earlier in this section do not encode enough information to obtain $total(u)$. To compute the desired values, we must rethink what our positions are and ensure that they encapsulate enough information to recover the total score.

In the new formulation, an anchor position is a 4-tuple $(y, U, upper, lower) \in \{null, 0, 50, 150, \ldots 1250\} \times (\mathcal{C} - \{Y\} \times \{0, \ldots, 63\} \times \{0, \ldots, 227\})$. y represents the state of the Yahtzee category and associated bonuses, U is the set of used categories (excluding Yahtzee), $upper$ is the upper total, and $lower$ is the lower total (plus any score in the upper categories in excess of 63). This representation was chosen for compactness of binary representation – the number of possible values for each component is close to a power of two; it takes 30 bits total to store them all. There are about 123 million reachable states.

Data for the optimal strategy are presented in Table IV.

TABLE IV
DISTRIBUTION OF SCORES FOR THE OPTIMAL STRATEGY

n	$P(score \geq n)$
50	$1 - 6.661782 \cdot 10^{-12}$
100	0.999998
150	0.991230
200	0.863584
250	0.483683
300	0.143265
400	0.038351
500	0.007192
750	$5.11603 \cdot 10^{-6}$
1000	$5.57508 \cdot 10^{-9}$
1250	$6.49213 \cdot 10^{-13}$
1500	$3.93308 \cdot 10^{-19}$

This information can be used to compute a strategy that beats the optimal solitaire strategy at an imperfect information version of two-player Yahtzee in which two players play a game simultaneously without knowing what the other is doing. In this version of the two-player game, the players reveal their scoresheets only once both have finished; the player with the higher score is the winner.

To compute a strategy to beat a given solitaire strategy at this version of the two-player game, we use the same definition of a game position as was used to compute the score distribution, and for each position compute the probability of beating the optimal solitaire strategy from that position. For a terminal position u with total score $total(u)$, the expected proportion of wins is

$$X_{win}(u) = \sum_{n < total(u)} P_{total,f}(n). \tag{10}$$

We work backwards in a manner similar to that described in Section II to compute the proability of winning from the non-terminal positions. The only computation that has to change is that for choosing a category: replace Equation 3 with

$$X_{win}(u) = \max_{(u,v) \in E} X_{win}(v) \tag{11}$$

(the term involving the score has been removed; the score is now encoded in the positions, so the expected win frequency $X_{win}(v)$ already takes the score into account). $X_{win}(s)$ for the initial position s then gives the optimal proportion of wins against strategy f. This technique is similar to that used to determine how to maximize the chance of beating a given score [6].

When playing against the optimal solitaire strategy, $X_{win}(s)$ is about 0.503.

III. Other Solitaire Strategies

The techniques of Section II-C can be applied to non-optimal strategies as well. This can aid us in developing and evaluating approximations to the optimal solitaire strategy. This can be useful for four reasons: 1) we can explore strategies that a human might be able to easily follow; 2) we can develop a sequence of increasingly good strategies in order to match human players of different abilities against appropriately able computer strategies; 3) we can examine strategies that do not require as much storage space or time to use in simulations (for the optimal strategy, one can keep the entire database of over half a billion position values, or one can keep just the primary 314,880 and take the time to recompute the others as needed); and 4) we hope to apply what we learn designing strategies for solitaire Yahtzee to the two-player perfect information version of the game (it is not likely to be feasible to compute the optimal strategy for the two-player game exactly because of the immense size of the position graph). The following sections explore some of these kinds of strategies.

A. Human Strategies

The first two strategies we describe are strategies that a human player could easily follow. The first is intended to reflect how a novice player might play: it sacrifices almost everything in order to obtain Yahtzee bonuses. To this end, it keeps whichever dice it has the most of, regardless of whether the corresponding upper category is open, and regardless of whether it is close to (or has) a straight or full house. When choosing a category to play a roll in, it seeks to maximize the score on that turn, with ties broken in favor of categories that are hard to get (for example, it will put a non-zero score in *Four Of A Kind* before *Chance*). When it cannot obtain a non-zero score in any category, it will zero the hard-to-get categories first, except that it saves *Yahtzee* until the end.

The second human-playable strategy is based on sets of rules that govern which dice to keep and what categories to choose; the first rule that applies is followed. Table V gives the rules for deciding which dice to keep. The first rule that matches is followed. Table VI determines which category to score a roll in; it is implicit that a rule does not match if the corresponding score would be zero. As for the keeps, the first rule that matches takes precedence. If no rule matches, the categories are examined in the order (1, C, 2, Y, 4K, LS, SS, 3, FH, 4, 3K, 5, 6); the first unused category is chosen.

A genetic algorithm can be devised to evolve strategies based on rules like those in Tables V and VI. In this setup a strategy is viewed as three lists: one list of rules that determine which dice to keep; one list of rules that determines what category to score a roll in; and one permutation of the 13 categories that is used when no rule in the second list matches. The rules on the first two lists are of the form $(c, score, O) \in \mathcal{C} \times \{0, \ldots, \ldots 50\} \times \mathcal{P}(\mathcal{C})$. A rule $(c, score, O)$ matches a position $v = (U, upper, extras, R)$ if $c \notin U$ (c is unused), R would earn more than $score$ points in c, and $O \cap U = \emptyset$ (all of the categories in O are unused as well). For the list of rules governing which dice to keep, the score of a roll for some of the lower categories can be defined according to how close the roll is to fitting that category. For example, [2 2 3 4 4] could be considered worth 10 points towards *Full House* at the beginning of a turn. Such parameters are currently considered part of the environment; they are not encoded in the population and are not evolvable.

TABLE V
Sample Rules for Keeping Dice

Rule
Yahtzee if Y is unused or Yahtzee Joker is applicable
Large Straight if LS or SS is unused
Small straight if SS is unused or both LS and C are unused
a tripleton if the corresponding upper category is unused
any tripleton if one of 3K, 4K, FH, or C is unused
a doubleton (high preferred) if the corresponding upper category is unused
[2 3 4] or [3 4 5] if SS unused or both LS and C are unused
any doubleton if 3K or C is unused
any tripleton (high preferred) if *Yahtzee* is unused or non-zero
a singleton (low preferred) if the corresponding upper category is unused, unless more than four upper categories are unused
any doubleton (high preferred)
a singleton 4, 5, or 6 (high preferred) if 3K, 4K, or C unused
nothing

TABLE VI
Sample Rules for Scoring Rolls

Rule
Yahtzee
Large Straight
Small Straight
a tripleton in an upper category if it earns the bonus
four 5's or four 6's in the upper category
Four of a Kind
three 5's or three 6's in the upper category
Full House
Three of a Kind if the total is at least 22
a tripleton in an upper category
Three of a Kind
Chance if the total is at least 22
doubletons in an upper category (lower preferred)

At a position $v \in V_2 \cup V_4$, the list of keep rules is consulted. The first matching rule is followed, and the dice are kept in an effort to make a good score in the category of that rule. For example, if the category in the matched rule is *Four of a Kind*, then the strategy keeps the dice is has the most of. For *Small Straight* it would keep the longest consecutive sequence. If no rule matches, all five dice are rerolled.

At positions $v \in V_6$, the list of scoring rules is consulted. The first rule that matches is followed by choosing the corresponding category. If no rule matches, the third list is consulted, and the first category on that list that has not been used is selected.

The initial population of strategies contains zero or one randomly generated rule on the keep list, and has on the scoring list a rule for each category that expresses "score in this category if the score would be non-zero"; these rules are randomly permuted. The permuation of categories on the third list is also randomly generated. Crossover is performed individually on the three lists. For the first two, the lists from the parents are shuffled together and half of the rules are removed from the result. For the third list, the position of each category is randomly chosen to be the same as in one of the parents. When there is a conflict, the category from the first parent is listed first. Any holes in the list are then closed up. Mutation is performed by either randomly swapping two items on one of the lists, or randomly changing, deleting, or adding an item on the first two lists.

The best strategy evolved using this scheme does not do as well as the list-based strategy descirbed above. This is perhaps not surprising considering the rules used are not expressive enough to encode all of the rules in Table V. Two encouraging things can be reported, however. First, the third list, which essentially determines the order in which categories are used when no good score can be made on a turn, evolves well. In one run, nearly all of the population had *Ones* first, with *Chance* and *Yahtzee* not far behind. This is similar to the optimal strategy's behavior [2]. Second, the second list (governing scoring) shows signs of evolving well too. The population learns the most obvious rules, such as "score in *Yahtzee* if doing so earns 50 points"; it is the more subtle rules that seem to be eluding us.

B. Strategies Based on Estimates of Position Values

As noted at the beginning of this section, when simulating an optimal solitaire player, we need only have access to the position values of the anchors; all of the other position values can be recomputed as necessary using the values at the anchors. This is a compromise between CPU time and storage space. If we keep all of the position values then looking up what the optimal strategy says to do is easy, but our database must hold half a billion position values (several gigabytes if the position values are stored in 8 bytes each). On the other hand, if we keep none of the values we must recompute everything every time we want to query the strategy. By keeping the partial database (several megabytes), we need only recompute the position values for the components that are visited in the course of a game.

We can think of the database of position values for anchor positions as a function $\hat{X} : V_1 \rightarrow \mathbf{R}$. We can perform the calculations of Equations 1-3 using $\hat{X}$ instead of X. If $\hat{X}(u) = X(u)$ for all anchors u, we get the optimal strategy. Using different $\hat{X}$ functions will yield different strategies. The point of using a different $\hat{X}$ is that we can devise them so they use very little storage. This could be useful in applications where storage space is extremely limited, such as handheld devices.

The simplest partial database $\hat{X}$ would be $\hat{X}(u) = 0$ for all anchors u. The corresponding strategy would then work to maximize the number of points earned on each turn without regard for the score in subsequent turns. We therefore call this strategy the *greedy strategy*.

1) Using the Expected Scores in Unused Categories: Better strategies can be obtained by using better heuristics to estimate the value that would be in the optimal partial database. One heuristic uses the statistics given in Section II-C. The expected future score for an anchor position $u = (U, upper, extras)$ is estimated to be

$$\hat{X}(u) = \sum_{c \in \mathcal{C}-U} \bar{s}(c); \tag{12}$$

adjustments can be made to estimate the probability of earning any bonuses. Those adjustments can be made more accurately if the variances for each category are used as well.

It should be noted that we can replace the averages $\bar{s}(c)$ in each category in Equation 12 with *any* function $s : \mathcal{C} \rightarrow \mathbf{R}$ to get

$$\hat{X}(u) = \sum_{c \in \mathcal{C}-U} s(c) \tag{13}$$

(for simplicity we will now ignore the adjustments for bonuses made using the variances). We can analyze the strategy obtained using Equation 12 using Algorithm 3 to obtain its averages in each category. Those averages can be used to define $s(c)$ in Equation 13 to obtain yet another strategy. This process can be iterated until we reach a fixed point or see sustained regression. The average total scores obtained by the first few strategies in this sequence are 225.40, 236.56, 232.27, 235.79. Of course, there is no guarantee that a fixed point will be reached, or that even if there is one, it maximizes the average total score over all strategies based on Equation 13.

What we want to do is find the vector of real numbers $(s(1), s(2), \ldots, s(Y))$ that maximizes an objective function (the average total score of the corresponding strategy). This is an appropriate problem for evolutionary algorithms. Some challenges to using evolutionary algorithms in this context include 1) the objective function takes a long time to compute exactly; and 2) estimating the objective function by simulation is extremely noisy (the standard deviation of the final score for the optimal strategy is 59.61). In addition, we suspect that the objective function is highly multi-modal and that maintaining diversity (or, in evolution strategy terms, balancing exploitation and exploration) will be essential and difficult.

Arnold and Beyer find that evolution strategies are more robust than other optimization algorithms in a simple environment with high levels of noise [7], however, efficiency still drops with increased noise. We need to run enough simulations to reduce noise to a managable level, but how far should we go? Fitzpatrick and Grefenstette suggest that in noisy environments and given a fixed number of function evaluations, it is better to have a larger population with fewer evaluations than a smaller population with more evaluations (and hence less noise) [8]. Beyer contradicts this. Jin and Branke survey more answers to this question, along with many other approaches to dealing with noise [9].

In our preliminary exploration of using evolutionary techniques to optimize s, we have used a simple genetic algorithm. Initial results suggest that, in our environment, increasing the population at the expense of more noise in the objective function is beneficial: nine test runs using a population of size 64 and sample size of 400 yielded an average best generation with a score of 238.52; using a population size of 256 and a sample size of 100 yielded an average of 241.51. This difference is statistically significant with $P = 0.008$. Intermediate population sizes (with sample sizes chosen to keep the total number of evaluations constant) yielded slightly lower averages (between 240.5 and 241.25), but the differences were not statistically significant. This suggests that there is a benefit to higher population sizes to some point; beyond that point there is little advantage.

The best strategy we have been able to evolve obtains an average final score of 243.63, which is better than any of the other strategies we have constructed manually based on this technique, including the strategy that gets additional information in the form of the variances (which was intended to give that strategy a better chance of earning the upper bonus). Further work will be done to optimize the parameters of our genetic algorithm to examine more complicated genotypes. For example, we could optimize the expected variances as well to better predict the upper bonus. Doing that for the manually created strategies yields a gain of over five points; if a similar gain is realized for our genetic algorithm champion then it will be very close to our best heuristic.

2) A Strategy Based on Simpler Games: For our final heuristic, imagine a version of Yahtzee with no lower categories and another version with no upper categories. These simple versions can be solved easily, and in fact the position values are already saved in the database for the complete game. Let $U \subseteq \mathcal{U}$, $0 \leq upper \leq 63$, $L \subseteq \mathcal{L}$, and $extras \in \{0, 1\}$. Then $(U, upper)$ is an anchor position in the former game and $(L, extras)$ is an anchor in the latter game. Now the maximum expected future scores for those positions are $X'(U, upper) = X(U \cup \mathcal{L}, upper, 0)$ and $X'(L, extras) = X(L \cup \mathcal{U}, 0, extras)$ since we can view the modified game that uses only the upper categories as a game that starts on the position graph for the complete game at the position where all of the lower categories are used and no Yahtzee bonuses can be earned. The case for the simpler game on only the lower categories is similar. We can then estimate the value of a position $u = (U, upper, extras)$ in the original game by

$$\begin{aligned} \hat{X}(u) &= X'(U \cap \mathcal{U}, upper) + X'(U \cap \mathcal{L}, extras) \\ &= X(U \cup \mathcal{L}, upper, 0) \\ &\quad + X(U \cup \mathcal{U}, 0, extras). \end{aligned} \tag{14}$$

Note that we are only using $2^6 \cdot 64 + 2^6 \cdot 3 = 4,288$ values of X.

C. Statistics for Non-optimal Strategies

Knowing the average score for each category (obtained from Algorithm 3) when following a given strategy allows us to compute the average total score for that strategy. The average scores for the strategies described above are given in Table VII.

TABLE VII
STATISTICS FOR NON-OPTIMAL STRATEGIES

Strategy	Average Score
Yahtzee Bonus	170.45
Rule-Based (from genetic algorithm)	189.75
Rule-Based (Tab. V, VI)	218.18
Greedy	218.54
Category Means Heuristic (Eq. 12)	225.40
Category Means Heuristic (Eq. 13)	236.56
Category Means Heuristic with Variances	241.94
Category Means Heuristic (from GA)	243.63
Upper and Lower Heuristic (Eq. 14)	250.41
Optimal	254.59

IV. CONCLUSION AND FUTURE WORK

We have presented improvements to the retrograde analysis of solitaire Yahtzee and techniques for analyzing strategies for solitaire Yahtzee. These techniques allow us to 1) determine the average score in each category and the variance of those scores; and 2) determine, for any n, the probability that the strategy will score n points. This gives us a method to evaluate approximations to the optimal solution and also allows us to compute strategies to win at an imperfect information version of the two-player game. This work leads to the following questions.

1) Can rule-based strategies be developed for solitaire Yahtzee that challenge the best heuristics or even the optimal strategy?
2) Can evolutionary techniques be used to futher improve on the current heuristics?
3) Can evolutionary techniques be used to design competitive strategies a human could follow?
4) Can heuristics be developed for the perfect information two-player game?

REFERENCES

[1] J. Glenn, "An optimal strategy for yahtzee," Loyola College in Maryland, Tech. Rep. CS-TR-0002, 2006.
[2] T. Verhoeff, "Solitaire yahtzee: Optimal player and proficiency test," http://svsoc1.win.tue.nl/ wstomv/misc/yahtzee/, 2001, visited October 30, 2006.
[3] F. Holderied, "Über das perfekte kniffel-spiel," http://holderied.de/kniffel/, 1999, visited October 30, 2006.
[4] P. Woodward, "Yahtzee: The solution," *Chance*, vol. 16, no. 1, pp. 18–22, 2003.
[5] J. Glenn, H. Fang, and C. P. Kruskal, "A retrograde approximate algorithm for one-player can't stop," in *5th International Conference on Computers and Games*, 2006, (to appear).
[6] C. Cremers, "How best to beat high scores in yahtzee: A caching structure for evaluating large recurrent functions," Master's thesis, Eindhoven University of Technology, June 2002.
[7] D. Arnold and H.-G. Beyer, "A comparison of evolution strategies with other direct search methods in the presense of noise," *Computational Optimization and Applications*, vol. 24, pp. 135–159, 2003.
[8] J. Fitzpatrick and J. Grefenstette, "Genetic algorithms in noisy environments," *Machine Learning*, vol. 3, pp. 101–120, 1988.
[9] Y. Jin and J. Branke, "Evolutionary optimization in uncertain environments – a survey," *IEEE Transactions on Evolutionary Computation*, vol. 9, no. 3, pp. 303–317, June 2005.