OPTIMIZING GENETIC ALGORITHM PARAMETERS FOR A STOCHASTIC GAME

James Glenn
Department of Computer Science, Loyola University Maryland, 4501 N. Charles Street, Baltimore, Maryland, U.S.A.

Keywords: Genetic algorithm, Parameter optimization, Stochastic games, Heuristics.

Abstract: Can't Stop is a jeopardy stochastic game played on an octagonal game board with four six-sided dice. Previous work generalized a well-known heuristic strategy for the solitaire game and attempted to optimize the parameters of the generalized strategy using a genetic algorithm (GA). There were two challenges in that optimization process: first, the stochastic nature of the game results in a very noisy fitness function; second, the fitness function is computationally expensive. In this work we continue the optimization process for the heuristic strategy by optimizing the GA: for a fixed number of fitness function evaluations, we investigate the effects of varying the GA parameters (in particular the population size and number of generations), which in turn affect the number of samples per individual and thus noise as well. We also examine different sampling schedules; our schedules are unique in that selecting the final champion is considered a schedulable phase. The GA parameters are first optimized on an easy-to-compute test function. The resulting GA parameters are effective on the original problem and as a result we obtain an improved heuristic strategy for Can't Stop.

1 INTRODUCTION

Can't Stop is a board game for two to four players with elements of both strategy and chance (Sackson, 2007). Simplified versions of Can't Stop (including solitaire versions) have been solved (Glenn et al., 2008), but the original game is difficult to solve because of the large number of states that must be evaluated. Keller (1986) developed a heuristic strategy that Glenn and Aloi later generalized into a parameterized strategy for solitaire Can't Stop, where the parameters were selected with a genetic algorithm (GA) (Glenn and Aloi, 2009); the fitness of a set of parameters was the average number of turns needed to complete the game using the corresponding strategy. That fitness function was estimated by repeated simulations of game play. Because the amount of time needed to simulate a game is nontrivial, it is not feasible to simply run the GA until the population stops improving; it is more practical to limit the number of simulations performed during each run of the GA. However, because of the noisy nature of the estimated fitness function, the GA is sensitive to its parameters, particularly the population size and number of generations, and to the scheduling of the evaluations (studied in a different domain by Aizawa and Wah (1993)), which together will determine the number of simulations performed for a particular set of strategy parameters.

In this work we first optimize the GA parameters by substituting an easy-to-compute test function. We consider primarily population size, number of generations, Aizawa and Wah's "between-generations scheduling", and a new aspect of scheduling we call "phase scheduling". We obtain better parameters for the generalized solitaire Can't Stop strategy by optimizing them with the resulting GA.

2 RULES OF CAN'T STOP

Can't Stop was invented by Sid Sackson and originally published by Parker Brothers in 1980 (it is currently published by Face 2 Face Games (Sackson, 2007)). Can't Stop is one of a class of games called *jeopardy stochastic games* (or, more specifically, *jeopardy dice games* when the stochastic element is supplied by dice) in which each player's turn is a sequence of stochastic events, some of which allow the player to make progress towards a goal, and some of which will end the player's turn immediately. After each incremental step towards the goal, players can choose to end their turn, in which case the progress made during the turn is banked and cannot be lost on a later turn. Players who press their luck and

Glenn J..
OPTIMIZING GENETIC ALGORITHM PARAMETERS FOR A STOCHASTIC GAME.
DOI: 10.5220/0003079101990206
In *Proceedings of the International Conference on Evolutionary Computation* (ICEC-2010), pages 199-206
ISBN: 978-989-8425-31-7

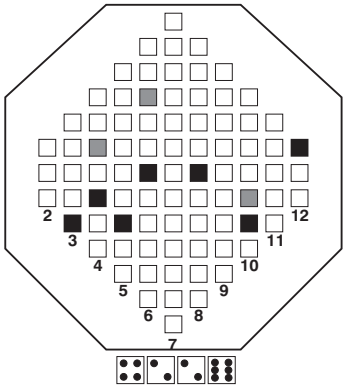

Figure 1: A Can't Stop position. Black squares represent the colored markers; gray squares are the neutral markers.

choose to continue their turns risk being forced to end their turns by an adverse outcome of the stochastic event (for example, rolling a one), in which case they lose any progress made during the turn. Pig (solved for two players (Neller and Presser, 2004; Neller and Presser, 2006)), Ten Thousand, and Cosmic Wimpout are other examples of jeopardy stochastic games.

The specific rules for Can't Stop are as follows. The game board has columns labelled 2 through 12 (the totals possible on two dice). Columns 2 and 12 are three spaces long, 3 and 11 are five spaces long, and so forth to the thirteen spaces in column 7. Each player has one marker for each column, colored to distinguish them from other players' markers. There are three neutral markers (white) that are used to mark progress during a turn. Each turn follows these steps:

(1) the current player rolls four six-sided dice;

(2) the player groups the dice into two pairs in such a way that progress can be made in the next step – if that is impossible then the turn ends immediately with the neutral markers removed from the board and the colored markers left as they are;

(3) in each column for the pair totals, either a new neutral marker is placed one space above the player's colored marker or the neutral marker is advanced one space, depending on whether there was already a neutral marker in the column;

(4) the player chooses between returning to step (1) or ending the turn, in which case the player's colored markers advance to replace the neutral markers.

The goal of the game is to be the first player to advance to the top of any three columns (or to do so in as few turns as possible for the solitaire version). Once a player wins a column then no player can make further progress in that column. The player must use both pair totals if possible, but is allowed to choose which to use if the pairing in step (2) results in pairs such that one or the other can be used, but not both at the same time (this can happen if only one neutral marker remains). (Note that there is ambiguity in the official rules: we interpret the rule "if you can place a marker on your roll, you must" so it applies *after* a player has chosen how to pair the dice.)

For example, in Figure 1 the possible pair totals would be 4 and 10 or 6 and 8. In the former case the neutral markers would be moved ahead one space in columns 4 and 10. In the latter case the marker in column 6 would be moved up one space but no progress would be made in column 8 because all three neutral markers have been used. If the roll had been 6-6-6-6 then the player would have lost all progress because the only possible pair totals would be two 12s, but column 12 has been won and so is out of play.

3 VALUE ITERATION AND NEWTON'S METHOD

Retrograde analysis is a common bottom-up technique used to compute game-theoretic values of positions by starting with the terminal positions and working backwards towards the starting position (Ströhlein, 1970). For acyclic games, retrograde analysis simply evaluates positions in reverse topological order. This technique has been used to solve solitaire Yahtzee (Woodward, 2003; Glenn, 2006).

Retrograde analysis in its more complex forms has been applied to endgames for non-stochastic games including chess (Thompson, 1986; Thompson, 1996), checkers (Lake et al., 1994; Schaeffer et al., 2004), and Chinese chess (Wu and Beal, 2001; Fang, 2005a; Fang, 2005b), and has been used to solve Nine Men's Morris (Gasser, 1996), Kalah (Irving et al., 2000), and Awari (Romein and Bal, 2003).

The cyclic and stochastic nature of Can't Stop requires a different approach. The cycles arise from the fact that a turn can end with no progress made. Value iteration is one approach to handling the cycles (Bellman, 1957). The value iteration algorithm starts with estimates of the position values of each vertex. Each vertex's position value is then updated (in no particular order in the most general form) based on the estimates of its successor's values to yield a new estimated value. In this way the estimates are refined until they converge.

Because the cycles in Can't Stop are only one turn long (progress that has been banked can never be lost), the game graph can be decomposed into components, each of which component consists of an *anchor* representing the start of a turn and all of the positions that can be reached before the end of that

turn. The components form an acyclic graph and can be processed in reverse topological order. Iterative methods can then be used within the components; one method (Glenn et al., 2008) breaks the cycles within the components by removing anchors' incoming edges, guesses a position value x for the anchor, and then computes a new estimate $f(x)$ of the anchor's position value. The resulting function f is continuous and piecewise linear; the fixed point of f gives the position value of the anchor. For any x, $f(x)$ and $f'(x)$ can be computed using retrograde analysis within the component. Computing $f'(x)$ allows the use of Newton's method, which converges to the fixed point significantly faster than value iteration or its variants.

4 HEURISTIC STRATEGIES

The method described above has been used to solve simplified variants of solitaire Can't Stop. These variants use dice with fewer than six-sides and a board with possibly shorter columns. The variants are referred to as (n,k) Can't Stop where n is the number of sides on the dice and k is the length of the shortest column (with adjacent columns always differing by 2 in length).

The most complex version of solitaire Can't Stop that has been solved is $(5,2)$ Can't Stop. Evaluating its 17 billion positions took 60 CPU days; an estimate for the time required to solve the official game using current techniques is 3000 CPU years. Heuristic strategies for the full game are therefore still of interest. Even for simple games heuristics are more useful to human players: no human can memorize the data or mentally perform the calculations needed to replicate the optimal strategy for $(5,2)$ Can't Stop.

4.1 The Rule of 28

One heuristic strategy is the Rule of 28 (Keller, 1986). The Rule of 28 is used to determine when to end a turn by assigning a *progress value* to each configuration of the neutral markers. Players should end their turn when this value reaches or exceeds 28. The progress value computation is split into two parts: one part for measuring the progress of the neutral markers; and one part for assessing the difficulty of making a roll that will allow further progress.

The first part of the progress value is computed as column-by-column sum. The value a column contributes to the sum is computed as some constant weight assigned to that column times one more than the number of spaces advanced in that column. The weights are one for column 7, two for columns 6 and 8, and so forth to six for columns 2 and 12, reflecting the fact that it is more difficult to make progress in the outer columns, and those columns are shorter, so progress in them is therefore more valuable. So if $\vec{s} = (s_2, \ldots, s_{12})$ where s_i is the number of spaces of progress in column i during the current move, then the marker progress value is

$$p_m(\vec{s}) = \sum_{i=2}^{12} (s_i + 1)(|7 - i| + 1). \qquad (1)$$

Because certain combinations of columns are riskier to be in than others, a difficulty score is added to that sum. For example, if a roll is all evens then it is impossible to make an odd pair total. Therefore, two points are added to the progress value when all three neutral markers are in odd columns. On the other hand, every roll permits at least one even pair total, so if the neutral markers are all in even columns, two points are *subtracted* from the progress value, lengthening a turn. Additionally, four points are added when the columns are all high (≥ 7) or all low (≤ 7).

For example, in Figure 1 the progress value for column 4 is $(2+1)\cdot 4 = 12$, the progress value for column 6 is $(3+1)\cdot 2 = 8$, and the progress value for column 10 is $(1+1)\cdot 4 = 8$. Because all three neutral markers are in even columns, 2 points are subtracted to get a total progress value of $12+8+8-2$ = 26. The Rule of 28 suggests rolling again.

A similar scheme can be used to determine how to pair the dice: each column is assigned a weight and each possible move is scored according to the weights of the columns it would make progress in. The total is called the *move value*; the move with the highest move value is the one chosen. Giving the outer columns lower weights than the middle columns (thus favoring choosing the middle columns) works better than the opposite pattern. In order to conserve neutral markers, a penalty is subtracted for each neutral marker used. In particular, if $\vec{p} = (p_2, \ldots, p_{12})$ where p_i is the number of squares advanced by a move in column i and $\vec{m} = (m_2, \ldots, m_{12})$ where m_i is 1 if the move places a new neutral marker in column i and 0 otherwise, then the total move value is

$$\mathrm{v}(\vec{p}, \vec{m}) = \sum_{c=2}^{12} (p_i \cdot (6 - |7 - i|) - 6 \cdot m_i) \qquad (2)$$

For example, in Figure 1 using the 6 has a score of 5 (the move value of one space in column 6). Using the 4 and 10 has a score of $3+3=6$, so this rule suggests using the 4 and 10.

When we henceforth refer to the Rule of 28 we mean the Rule of 28 combined with the above method

of choosing how to pair the dice. This strategy averages approximately 10.74 turns to win the solitaire game.

4.2 Generalizing the Rule of 28

Any of the constants assigned to the columns can be altered, as can the threshold and any of the difficulty values. Furthermore, the spaces within a column needn't be assigned the same weights. In general, to evaluate a particular move we denote the move by two vectors $\vec{m} = (m_2, \ldots, m_{12})$ and $\vec{n} = (n_2, \ldots n_{12})$ where m_i is the position of the neutral marker in column i (or the colored marker if there is no neutral marker) before the move and n_i is the position the neutral marker would advance to after the move. Assign the weight x_{ij} to space j in column i. Then the total move value v is the sum of the weight of the spaces that would be advanced over in the current turn if that move was made:

$$\mathrm{v}(\vec{m}, \vec{n}) = \sum_{i=2}^{12} \sum_{j=m_i+1}^{n_i} x_{ij}. \quad (3)$$

The same technique could be applied to progress values as well.

4.3 Linear Weights Strategies

Previous work studied two constrained versions of the generalized heuristic (Glenn and Aloi, 2009). The most flexible (and most successful) constrained the progress values to be constant within each column and required the move weights x_{ij} to be described by a linear function within each column. Specifically, a *Linear Weights strategy* was described by $(p_2, \ldots, p_7, m_2, ..., m_7, b_2, ..., b_7, e, o, h, k, t)$ where

1. the p_i are progress values with $p_i \in \{0, \ldots 7\}$ for each column $i \in \{2, \ldots, 7\}$ (symmetry is used so that, for example, $p_8 = p_6$),
2. m_i and b_i define the linear function for column i that determines the move weights of each space within that column:

$$x_{ij} = \left\lfloor m_i \cdot \frac{j}{l_i} + b_i \right\rfloor \quad (4)$$

   (where l_i is the length of column i, m_i is chosen from 32 somewhat arbitrarily chosen values between 0 and 64, and $b_i \in \{0, \ldots, 7\}$),
3. e, o, and h are the even, odd, and high penalties (each chosen from between -8 to 7 and with the low penalty l equal to the high penalty h),
4. k is the marker penalty (between 0 and 15), and
5. t is the progress threshold (between 0 and 31).

The best Linear Weights strategy found achieved an average score of 9.05 turns, with a standard deviation of 2.30.

5 GENETIC ALGORITHM

The parameters of the Linear Weights strategies have been optimized using a genetic algorithm (GA) (Glenn and Aloi, 2009). Each candidate strategy was encoded using the appropriate number of bits for each parameter (87 total). Various-sized populations of those bit strings were subjected to standard GA operators (two-point crossover, mutation, two-round tournament selection) for twenty generations. The fitness of an individual strategy was taken to be the expected number of turns that strategy takes to finish the solitaire game. That expectation was estimated by computing the mean over n simulated games. The high level of noise in the evaluation function (the best strategy found has a standard deviation of approximately 25% of its mean turns to complete the game) was therefore an issue. Some attempt was made to determine the optimal population size for a fixed number of total evaluations, but the results were not conclusive. We now strive to better optimize the parameters of the Linear Weights strategies; to do so we will first optimize the GA parameters. We continue with a GA instead of some other optimization method for two reasons:

1. the fitness function is multi-modal; and
2. Arnold and Beyer (2003) found that in a simple environment with high levels of noise, evolution strategies were more robust than other optimization algorithms, and we assume that this robustness is shared with genetic algorithms.

5.1 Optimizing Genetic Algorithm Parameters in the Presence of Noise

For a fixed number of total simulations there is a tradeoff between using more evaluations per individual (thus reducing noise) and having a larger population (increasing diversity and allowing more of the search space to be examined). Fitzpatrick and Greffenstette (1988) performed analysis suggesting 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. They then presented empirical work with two problems (a noisy version of one of De Jong's (1975) original test functions, and medical image registration) that confirmed their analysis to

a point: choosing a population size that allowed two evaluations per individual was generally better than choosing a population size that allowed only one evaluation. They used a fixed number of generations, except that they reduced the number of generations in cases where it was necessary to compensate for the additional GA overhead required for larger populations (that is, when the GA overhead was not dominated by the total evaluation time).

Arnold and Beyer (2003) found that for low levels of noise, efficiency drops as population size increases. These two results together suggest that there is an optimal population size that is reached once noise is reduced sufficiently. Experiments with the solitaire version of the game Yahtzee (Glenn, 2007) presented some confirmation of that, although the value examined was the average fitness in the final generation rather than the best *individual*.

Aizawa and Wah (1993) considered the scheduling of a fixed number of evaluations for a fixed population size in two senses: "duration scheduling" or "between-generation scheduling", which determines how many evaluations are used during each generation; and "sample allocation" or "within-generation scheduling", which determines how many evaluations are allocated to each member of a population within a single generation. They found that performance improves when more evaluations are scheduled in later generations, and that a dynamic approach to within-generation scheduling offers further improvements.

Jin and Branke (2005) survey more approaches to dealing with noise.

We now wish to consider more completely the effects of GA parameters on the results with the aim of further optimizing the Linear Weights strategies. However, since a single run of the GA can take several hours, it is computationally expensive to do so by trial and error (or by using a meta-GA). Instead, we optimize the GA parameters for a function that is easier to compute and then hope it has characteristics similar enough to those of the fitness function for the Linear Weights strategies so that the same GA parameters will work well for the latter.

5.2 Schwefel's Function

The function we choose to model noise is Schwefel's function, a well-known function used to benchmark genetic algorithms (Mühlenbein et al., 1991; Törn and Zilinskas, 1989). Schwefel's function takes n real numbers as inputs and is defined by

$$s(x_1,...,x_n) = \sum_{i=1}^{n} 420.9697 - x_i \cdot \sin\sqrt{|\,x_i\,|}. \quad (5)$$

Schwefel's function is continuous and highly multimodal; those properties along with the fact that it can be extended to any number of inputs makes it a desirable test case. We define a family of functions equal to Schwefel's function with noise introduced: let $s'_{n,\sigma}$ be defined by

$$s'_{n,\sigma}(x_1,...,x_n) = s(x_1,...,x_n) + \mathrm{N}(0;\sigma), \quad (6)$$

where the last term denotes a random variable with the given normal distribution. This variable is chosen anew each time $s'_{n,\sigma}$ is evaluated.

When the inputs are restricted to $[-500,500]$, Schwefel's function is minimized to zero when all inputs x_i are equal to $420.9697\ldots$ (which is near a boundary and far from the next-best local minima, which occur when $x_i = -302.5249\ldots$ for one i and all other x_i equal $420.9697\ldots$). In the absence of noise there are methods that have no trouble finding this minimum. We wish to determine, for different noise levels, the optimal parameters for a GA that uses a fixed number of evaluations of $s'_{n,\sigma}$. We will investigate how those parameters depend on n and σ, and how effective the resulting GA is.

5.3 Selecting GA Parameters

We now consider optimizing the following genetic algorithm parameters: 1) the population size; 2) the number of generations; 3) the between-generations schedule, simplified so the number of evaluations used during a particular generation is a linear function of the generation number; and 4) the "phase schedule". The phase-scheduling problem is a third type of scheduling problem we introduce to go along with between- and within-generation scheduling. Our GA is divided into two phases: the evolution phase, and the final champion selection phase. The latter phase is necessary for two reasons: because the expected number of evaluations per member of the population will be very low, it is impossible to select the overall most fit individual with any confidence; and it is currently infeasible to compute the exact fitness of a particular Linear Weights strategy. The latter complication is not present when dealing with test functions with artificially introduced noise such as Schwefel's function or De Jong's function (one can simply evaluate the exact fitness by not adding in the noise), and is not present to the same extent when dealing with Fitzpatrick and Greffenstette's problem of medical image registration (instead of estimating fitness by using a sample of pixels, one can compute the exact fitness by examining *every* pixel; there is additional overhead, but that overhead can be accounted for).

We therefore add a second phase to the GA after the final generation has been generated. In this phase

we use many more evaluations in order to greatly improve our fitness estimates so we can determine with some confidence the overall most fit individual (the "final champion") from among some pool of candidates, which we take to be the entire final generation.

We use a constant mutation rate and crossover rate and do not consider the effects of dynamic within-generation scheduling, although we do use a somewhat simplified version of dynamic scheduling in the final champion selection phase.

Results of a meta-GA used to determine optimal parameters for a GA optimizing $s'(x_1,\ldots,x_{10},1000)$ indicated that evaluation scheduling is less important than the population size and number of generations – the meta-GA tended to settle on the same values for population size and number of generations but chose wider ranges for the other parameters over several runs. We therefore examined the former two parameters more closely. We varied the number of generations between 32 and 1024 and the population size between 64 and 2048. We ran a GA with those parameters to optimize $s'_{n,\sigma}$. Each run of the GA used 2,500,000 total fitness evaluations, the same number of evaluations for each generation, and 16% of evaluations to select the final champion. Sixteen bits were used to represent each input to $s'(n)$. The mean fitness of the final champion over 100 runs of the GA for $s'_{10,1000}$, $s'_{10,4000}$, and $s'_{20,1000}$ are given in Tables 1-3. Values that are statistically significantly different than the minimum (boxed) at the $p = 0.05$ level are given in bold.

Table 1: Average fitness of champions for $s'_{10,1000}$.

	Population				
Gen.	128	256	512	1024	2048
32	**190.3**	**99.02**	**72.21**	**69.31**	**70.71**
64	**151.4**	**58.24**	**46.06**	**50.75**	**54.89**
128	**125.3**	**48.54**	40.79	**45.92**	**50.05**
256	**134.1**	**44.77**	40.30	**46.39**	**52.69**
512	**149.0**	**50.74**	41.48	**45.73**	**57.61**
1024	**113.5**	**54.95**	**47.23**	**51.50**	**59.51**

Table 2: Average fitness of champion for $s'_{10,4000}$.

	Population				
Gen.	128	256	512	1024	2048
32	**271.2**	**173.0**	**165.7**	**172.6**	**218.7**
64	**220.0**	**125.4**	**101.0**	**104.2**	**132.3**
128	**201.9**	**103.1**	91.78	89.89	**103.4**
256	**218.7**	**107.3**	89.74	93.67	**100.3**
512	**217.1**	**120.8**	**100.1**	94.85	**110.8**
1024	**240.4**	**138.5**	**109.7**	**105.1**	**114.8**

Table 3: Average fitness of champion for $s'_{20,1000}$.

	Population				
Gen.	128	256	512	1024	2048
32	**1115.**	**685.3**	**491.7**	**394.2**	**338.7**
64	**667.4**	**311.3**	**165.1**	**139.6**	**138.4**
128	**560.1**	**222.2**	**112.7**	103.4	**113.4**
256	**496.6**	**206.2**	100.1	101.7	**114.4**
512	**475.7**	**202.3**	106.6	106.1	**119.2**
1024	**446.3**	**203.7**	**119.4**	**117.0**	**131.6**

We can compare the performance of the GA in the presence of noise to the results obtained by the Breeder Genetic Algorithm (BGA) (Mühlenbein and Schlierkamp-Voosen, 1993). In the absence of noise, BGA was able to find the minimum of the 20-input version of Schwefel's function using 16,100 function evaluations. We performed roughly 155 times as many evaluations. Applying those evenly to the 16,100 inputs evaluated by BGA would reduce the noise level for $s'_{20,1000}$ to approximately $\frac{1000}{\sqrt{155}} \approx 80$; our GA on average found solutions about 1.25 standard deviations from the minimum.

The optimal parameters seem to be fairly insensitive to the noise level and the number of inputs. If anything, there is possibly a trend towards a larger population as noise and number of inputs increase. That is perhaps a counterintuitive notion since those changes would result in fewer evaluations per individual and one might expect that noise would be best mitigated by *increasing* the number of evaluations per individual.

6 TRANSFER TO CAN'T STOP

Guided by the results on the noisy version of Schwefel's function, we ran a GA to optimize the Linear Weights strategies using a population of 512 individuals evolved over 256 generations. We used 10^6 evaluations per execution of the GA, allocating an equal number of evaluations to each generation and 16% of the total to select the final champion from the last generation. Then, to check whether those parameters were optimal for the Linear Weights strategies, we reran the GA for populations of 128, 256, 512, and 1024 evolved over 64, 128, 256, and 512 generations. We ran the GA up to 60 times for each combination of parameters; the average fitness of the final champions are given in Table 4.

We have also run the GA with different between-generation schedules and different phase schedules to confirm that the parameters have the same (very small) effect on the GA for the Linear Weights strate-

Table 4: Effects of GA parameters on linear weights champions.

	Population				
Gen.	64	128	256	512	1024
64	**9.01**	**8.93**	**8.95**	**8.97**	**9.00**
128	**8.98**	**8.92**	8.90	**8.95**	**8.95**
256	**8.93**	8.91	8.89	**8.92**	**8.97**
512	**8.94**	8.91	**8.91**	**8.94**	**9.05**
1024	**8.96**	**8.92**	**8.95**	**9.01**	

gies that they do on the noisy Schwefel's function. Results are given in Table 5 and compared to the standard of 256 generations, population 256, 16% of evaluations to select the final champion, and constant between-generation schedule. Statistically significant differences are in bold; it seems that varying these parameters has as much effect as varying the population size or number of generations by a factor of two.

Table 5: Effects of evaluation scheduling on linear weights champions.

Variation	Champion Mean
Standard	8.889
8% to select champion	8.897
32% to select champion	**8.904**
3x evaluations in first gen.	**8.913**
3x evaluations in final gen.	8.890

The best strategy found over all the runs completes solitaire Can't Stop in an average of 8.78 turns (versus 9.05 for the best in the previous work by Glenn and Aloi (2009)). It was found using a GA with a population of 512 and 256 generations, and with a between-generations schedule that allocated three times as many evaluations to the final generation as to the first. Its parameters are given in Table 6.

Table 6: Overall linear weights champion.

Column	Progress	Move
2,12	15	$72x+15$
3,11	13	$48x$
4,10	8	$56x+3$
5,9	8	$44x$
6,8	4	$28x+7$
7	3	$32x+5$
Difficulty Scores		
odds		11
evens		-1
highs, lows		4
marker		12
threshold		61

The encoding used here uses one more bit per parameter, thus allowing a wider range of values for each. Furthermore, one bug has been fixed since Glenn and Aloi's original implementation of the generalized heuristic. To isolate the effects of those changes from the effects of the optimized GA parameters, we ran the GA with the changes in effect but with the old, unoptimized GA parameters (population 400, 20 generations, but 25% more evaluations). The average fitness of the final champion was 9.12 with a best over 40 runs of 8.93. About half of the improvement is therefore due to the more expressive encoding combined with the bug fix and about half is due to the optimized GA parameters.

6.1 Other Games

In a previous study (Glenn, 2007) we introduced a class of strategies for solitaire Yahtzee that use an estimate of the expected score in each category to estimate the position value at the start of each turn. We optimized the estimates using 34 runs of a genetic algorithm and found a strategy with an expected score of 243.63. We reran the GA 40 times using the same parameters as for Can't Stop and found a strategy with an expected score of 244.11, which closes 4.4% of the gap to the optimal score of 254.59.

7 CONCLUSIONS

We have confirmed previous results in other domains suggesting that, in the presence of noise, there is some fairly low optimal number of samples per individual. A key difference is the definition of "fairly low": Fitzpatrick and Greffenstette (1988) found the optimal value to be 2; our results show a optimal value between 10 and 20 for Schwefel's function and the parameterized Can't Stop heuristic. It remains to be determined what characteristics of the objective function and noise determine the optimal number of samples and other GA parameters.

Our study shows that optimal GA parameters for the generalized heuristic and for Schwefel's function are similar. The population size and number of generations were the most important parameters. The effects of the allocation of evaluations within and between phases (including our new final champion phase) is roughly equivalent to the effect of varying the population size or number of generations by a factor of two.

Finally, we were able to find a strategy for solitaire Can't Stop that performs better than the best previously known, which had been found using unoptimized GA parameters. The success was matched for a parameterized non-optimal solitaire Yahtzee strategy.

ACKNOWLEDGEMENTS

James Glenn was supported by sabbatical leave granted by the Loyola College of Arts and Sciences at Loyola University Maryland.

REFERENCES

Aizawa, A. and Wah, B. (1993). Scheduling of genetic algorithms in a noisy environment. In Forrest, S., editor, *Proc. 5th Intl. Conf. on Genetic Algorithms*, pages 48–55, San Mateo, CA. Morgan Kaufman.

Arnold, D. and Beyer, H.-G. (2003). A comparison of evolution strategies with other direct search methods in the presense of noise. *Comp. Opt. and App.*, 24:135–159.

Bellman, R. E. (1957). *Dynamic Programming*. Princeton University Press, Princeton, NJ, USA.

De Jong, K. (1975). *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*. PhD thesis, University of Michigan.

Fang, H. (2005a). The nature of retrograde analysis for Chinese chess, part I. *ICGA Journal*, 28(2):91–105.

Fang, H. (2005b). The nature of retrograde analysis for Chinese chess, part II. *ICGA Journal*, 28(3):140–152.

Fitzpatrick, J. and Grefenstette, J. (1988). Genetic algorithms in noisy environments. *Machine Learning*, 3:101–120.

Gasser, R. (1996). Solving nine men's Morris. *Computational Intelligence*, 12:24–41.

Glenn, J. (2006). An optimal strategy for Yahtzee. Technical Report CS-TR-0002, Loyola College in Maryland, 4501 N. Charles St, Baltimore MD 21210, USA.

Glenn, J. (2007). Computer strategies for solitaire Yahtzee. In *IEEE Symp. on Comp. Intell. and Games*, pages 132–139.

Glenn, J. and Aloi, C. (2009). A generalized heiristic for can't stop. In *Proc. 22nd FLAIRS Conf.*, pages 421–426. AAAI Press.

Glenn, J., Fang, H., and Kruskal, C. P. (2008). Retrograde approximate algorithms for some stochastic games. *ICGA Journal*, 31(2):77–96.

Irving, G., Donkers, J., and Uiterwijk, J. (2000). Solving Kalah. *ICGA Journal*, 23(3):139–147.

Jin, Y. and Branke, J. (2005). Evolutionary optimization in uncertain environments – a survey. *Evolutionary Computation, IEEE Transactions on*, 9(3):303–317.

Keller, M. (1986). Can't stop? Try the rule of 28. *World Game Review*, 6. See also http://www.solitairelaboratory.com/cantstop.html last visited Nov. 22, 2008.

Lake, R., Schaeffer, J., and Lu, P. (1994). Solving large retrograde analysis problems using a network of workstations. In van den Herik, H., Herschberg, I. S., and Uiterwijk, J., editors, *Advances in Computer Games VII*, pages 135–162. University of Limburg, Maastricht. the Netherlands.

Mühlenbein, H. and Schlierkamp-Voosen, D. (1993). Predictive models for the breeder genetic algorithm. *Evolutionary Computation*, 1:25–49.

Mühlenbein, H., Schornisch, M., and Born, J. (1991). The parallel genetic algorithm as function optimizer. *Parallel Computing*, 17:619–632.

Neller, T. and Presser, C. (2004). Optimal play of the dice game Pig. *The UMAP Journal*, 25(1):25–47.

Neller, T. and Presser, C. (2006). Pigtail: A Pig addendum. *The UMAP Journal*, 26(4):443–458.

Romein, J. W. and Bal, H. E. (2003). Solving the game of Awari using parallel retrograde analysis. *IEEE Computer Society*, 36(10):26–33.

Sackson, S. (2007). *Can't Stop*. Face 2 Face Games, Providence, RI, USA. Boxed game set.

Schaeffer, J., Björnsson, Y., Burch, N., Lake, R., Lu, P., and Sutphen, S. (2004). Building the checkers 10-piece endgame databases. In van den Herik, H., Iida, H., and Heinz, E., editors, *Advances in Computer Games 10. Many Games, Many Challenges*, pages 193–210. Kluwer Academic Publishers, Boston, USA.

Ströhlein, T. (1970). *Untersuchungen über kombinatorische Spiele*. PhD thesis, Fakultät für Allegemeine Wissenschaften der Technischen Hochschule München, Munich.

Thompson, K. (1986). Retrograde analysis of certain endgames. *ICCA Journal*, 9(3):131–139.

Thompson, K. (1996). 6-piece endgames. *ICCA Journal*, 19(4):215–226.

Törn, A. and Zilinskas, A. (1989). *Global Optimization*. Springer Verlag, New York.

Woodward, P. (2003). Yahtzee: The solution. *Chance*, 16(1):18–22.

Wu, R. and Beal, D. (2001). Fast, memory-efficient retrograde algorithms. *ICGA Journal*, 24(3):147–159.