

The window algorithm

D.A. Newlands & D. Brain
Deakin University, Australia

Abstract

Genetic algorithm (GAs) have long been known as effective search techniques for high dimensional spaces. Function finding attempts to discover a mathematical function that adequately describes a set of data. A standard genetic algorithm searches for all coefficients of a function concurrently, but there are difficulties in simultaneously minimising run times and maximising accuracy. One recent improvement to GA performance is Delta Coding which searches all coefficients in parallel but increases the accuracy as the search progresses. This paper presents another method of tackling the accuracy and speed trade-offs in GAs. Instead of searching for all coefficients at the same time, a sliding window of coefficients is searched. Experiments are performed into fitting functions to data points and results indicate that good performance can be obtained with small windows. The results indicate that the window algorithm is a useful modification to the standard GA. Some insight into window sizing is presented.

1 Introduction

Many mathematical techniques exist for fitting functions to data, such as collocation and osculation [Scheid, 1968] but, for example, when the domain is of very high dimensionality, when attempting to fit a non-linear function to spaces between data points or when the surface is discontinuous, mathematical techniques become computationally infeasible. It has been shown that some artificial neural networks [Ginsberg, 1993, Hecht-Nielsen, 1987] - a technique based on mathematical principles - cannot find some polynomials [Cardell et al., 1994]. Genetic algorithms [Goldberg, 1989] can search large spaces quickly and, due to their abstraction of data, have a wide

range of application, including function finding. Functions can be in any general form and modifications to the simple genetic algorithm such as Gray coding [Caruna and Schaffer, 1988, Srinivas and Patnaik, 1994], Delta Coding [Whitley et al., 1991] and real parameter operators [Goldberg, 1989, Janikow and Michalewicz, 1991, Wright, 1993, Michalewicz and Janikow, 1991] improve the effectiveness of genetic algorithms. A problem in the use of GAs is balancing efficiency of search against accuracy of solution since the more variables which are encoded, the longer the GA takes to complete the search. Searching for real coefficients of a polynomial of order n requires converging $n + 1$ real numbers at the same time. Alternatively, the GA could be run $n+1$ times converging one coefficient each time. It is well-known that the latter method is much faster. If b bits are used to represent each of c coefficients, then the first approach searches 2^{b*c} schemas while the alternative searches only $c * 2^b$ schemas. However, searching each coefficient separately may produce a poor result, as interactions between coefficients cannot be explored. This paper investigates the possibility of exploring a small group of coefficients, in a sliding window, at a time. This may provide a method of speeding up the algorithm while not losing accuracy by not investigating interactions between coefficients.

2 Delta coding

Delta coding [Mathias and Whitley, 1994a, Mathias and Whitley, 1994b] is an approach to the local minima problem. When applied to function finding problems, it has the ability to alter the precision of the search as it proceeds. The standard genetic algorithm is likely to quickly find a near optimal solution. This can be used as the basis for a more precise search.

Delta coding is performed in two stages. The first stage is a single iteration of a standard genetic algorithm terminating on convergence of any parameter in the population. The best solution found is saved as an *interim solution*. Stage 2 consists of iterations in which the delta parameters are altered. These are summed with the corresponding parameters of the interim solution, producing a combined solution as the input to the objective function. When convergence within any parameter of the current population is detected, the combined solution with the highest fitness is saved as the next interim solution. What makes Delta coding powerful is that the number of bits used to represent parameters changes after each delta iteration. This constantly alters the size of the GA's search space. Delta coding is an effective method for finding optimal solutions.

3 The window algorithm

An alternative approach to searching all coefficients at once is to search one coefficient at a time, running the GA multiple times. A method that allows

interaction between coefficients while requiring a smaller string length is desirable. Delta coding restricts its search to a smaller area by increasing the precision and restricting the range of coefficients in the search but this limits its capability of searching interactions between coefficients. An algorithm which restricts the GA string size but which explores interactions between coefficients has been devised. This algorithm approximates coefficients in groups using a sliding window.

The question of which subset of coefficients to use at a particular time presents itself. It is plausible that the first subset to be explored should contain the coefficients which affect the function most. It is expected that coefficients corresponding to higher orders of a function are more important than coefficients of lower orders. It is plausible that the first subset to explore should contain the highest order coefficients of the function. Once the best values of the coefficients have been found for that subset, the subset can be altered to exclude some coefficients and include other, as yet unexplored, coefficients by sliding a window over the coefficients. This is the higher-order-first (HOF) approach. Two problems arise from this method: the initial highest order is chosen arbitrarily and, if the domain of the function is $(-1, 1)$, higher order coefficients do not dominate.

Both of these problems could be solved by making the initial window contain the lowest order coefficients and move upwards to include higher order coefficients. The GA can be terminated when including higher orders into the discovered function does not improve the fit of the function to the data. This lowest-order-first (LOF) approach may allow the lowest order function that fits the data to be found.

3.1 Pseudo-code for LOF

The pseudo-code for the LOF approach to the Window Algorithm is shown below.

```
Set the window to include the lowest coefficients
Initialise array for saving best coefficients
WHILE not done
    WHILE the population has not converged
        Perform selection, crossover, and mutation
        FOR each string in the population
            Evaluate using coefficients currently represented
                in the window and coefficients previously saved
            IF string is better than the current best solution
                THEN
                    Save the lowest order coefficient represented
                        in string to the array of best coefficients
                END IF
        NEXT string
```

```

END WHILE
IF the highest order coefficient has not yet been
    included in the window
THEN
    Move the window to along one position to exclude
    the lowest order coefficient in the window and
    include the next higher order coefficient
ELSE
    done = true
END IF
END WHILE
    
```

The algorithm can be modified to perform a HOF search by replacing *lowest* with *highest* and vice versa . The algorithm was implemented using the GAC genetic algorithm package. This is a publicly available package created by William M. Spears of the US Navy Center for Applied Research in Artificial Intelligence.

4 Experiments

To compare various settings of the algorithm, experiments were carried out finding the coefficients of polynomials. These are simple functions amenable to analytic solution, however this is unimportant in evaluating various settings. The experiments compare the HOF and LOF approaches and various window sizes. The parameters are represented as 32-bit binary parameters with Gray coding. Data sets consisted of 25 random numbers for the chosen polynomial of order 20. Care was taken that the coefficients used to generate the data were within the valid range of parameters used in the GA and, hence, the generating coefficients can potentially be found.

The objective function used in these experiments characterised the error of a string using the sum of squares method:

$$sumofsquares = \sum_i^n (y'_i - y_i)^2$$

where n is the number of points in the data set. The GAC package requires that each string return a fitness value within the range $[0, 1]$, where a higher fitness equates to a better performing string.

The fitness function was chosen to be

$$Fitness = \begin{cases} 1 + (-error * THRESHOLD)/best_result & \text{if error} < best_result \\ THRESHOLD & \text{if error} = best_result \\ (THRESHOLD/(error - best_result + 1)) & \text{if error} > best_result \end{cases}$$

where $best_result$ is the smallest error so far found in any previous generation, and $THRESHOLD$ is the highest fitness a string can be given if it

does not improve on previous solutions. This function allows the population to converge as small improvements in the objective function translate to significant improvements in the fitness function. As generations increasingly improve, smaller differences in the objective function are represented by larger differences in the fitness function.

All trials used one-point crossover at a rate of 60%. The mutation rate was 0.5%. Trials were run one hundred times for each combination of window and population size.

Two polynomials with different coefficient domains and restricted function domains were chosen for the initial experiments. The GA was run using the HOF and LOF approaches of the Window Algorithm, as described above. The GA was also run without using the Window Algorithm, searching all coefficients at once to provide a baseline against which to compare the performance of the Window Algorithm. The object of these experiments was to determine how well the Window Algorithm performs relative to a standard GA. All experiments had a ceiling of x^{20} for the highest order permitted in the polynomial. This allowed the relative performance of the LOF approach to be compared to the HOF approach without having to focus on exploring termination conditions.

For each experiment, the polynomial used, domain of the function, and allowable coefficient range of parameters is shown. The decimal number on the first line of each cell is the average error of the best set of coefficients found by the GA over the hundred trials for that combination of window and population size. The set of numbers on the second line before the colon represents the tests with the same population size that the current test performed statistically significantly better than.

In all tests in this work statistical significance was measured at the $p < 0.05$ level using a 2-tailed t-test. The second set of numbers represents the tests with the same window size that the current test performed statistically significantly better than. A “-” indicates that test did not perform better than any other test in that category. It should be noted that a larger population performs a more exhaustive search. Therefore, it may be expected that as population size increases, error decreases.

4.1 Example 1

Polynomial: $0.25x^{18} + 0.9x^{17} - 0.3x^{16} + x^{15} + 0.54x^{14} - 0.9x^{13} + 0.6x^{10} - 0.8x^9 - 0.93x^8 + 0.44x^7 - 0.99x^6 + 0.31x^5 - 0.5x^4 - 0.28x^3 - 0.8x^2 + x - 1$
 Function Domain: $[-1, 1]$ Parameter Range: $[-1, 1]$

4.1.1 HOF

Table 1	Population Size				
Win Size	20	50	100	150	Mean
1	17.27 7,9:-	17.27 :-	17.16 :-	17.16 :-	17.22
3	17.03 7,9:-	13.33 5,7,9:20	12.88 1:20,50	12.68 1:20,50	13.98
5	17.17 7,9:-	14.25 1:20	12.65 1:20,50	11.52 1,3:20,50,100	13.90
7	19.28 :-	14.43 1:20	12.36 1,3:20,50	11.07 1,3,5:20,50,100	14.23
9	20.32 :-	14.04 1:20	12.43 1:20,50	10.39 1,3,5,7:20,50,100	14.30
all	49.11 :-	38.92 -:20	32.29 -:20,50	27.47 -:20,50,100	36.94

Table 1 shows the average error for each combination of population and window size for the HOF approach on Polynomial One.

For population size 20, the smaller window sizes of 1, 3, and 5 perform best. For population size 50, the small window size of 3 produces the best performance. For population size 100, the larger window size of 7 outperforms all others. For population size 150, window size 9 clearly is the best. For any fixed window size, as the population size increases, error decreases. Each window size (except 1) performs statistically significantly better as population size increases. Although window size 1 has a slight performance improvement from population size 50 to size 100, it is not statistically significant.

These results clearly indicate a correlation between optimal window size and population size. For all population sizes, an optimal window size seems to exist. As the window size departs from this optimum, error also increases. For window sizes higher than the optimum, this can be presumed to continue until a window size of 21 is reached (which is effectively the same as the basic GA). This experiment indicates that smaller populations perform best with smaller windows, and larger populations perform best with larger windows. When compared to the all-coefficients-at-once experiment, the HOF approach performs extremely well. The average error for all combinations of window and population size is smaller than even the best average error of the baseline GA.

4.1.2 LOF

Table 2	Population Size				
Win Size	20	50	100	150	Mean
1	14.42 5,7,9:-	16.97 :-	15.31 :-	14.50 :-50	17.22
3	15.40 5,7,9:-	11.22 1,7,9:20	10.59 1,5:20	10.26 1:20,50	13.98
5	19.18 :-	12.62 1,9:20	11.58 1:20	10.65 1:20,50,100	13.90
7	21.07 :-	12.80 1,9:20	10.38 1,5:20,50	10.26 1:20,50	14.23
9	26.25 :-	16.41 :-20	11.23 1:20,50	9.81 1,5:20,50,100	14.30
all	49.11 :-	38.92 :-20	32.29 :-20,50	27.47 :-20,50,100	36.94

Table 2 shows the average error for each combination of population and window size for the LOF approach on Polynomial One. These results show that for population size 20, smaller window sizes of 1 and 3 produce the least error. For population size 50, window size 3 performs best. Window size 7 performs best for population 100, although window size 3 is not significantly worse. For population size 150, window size 9 is clearly the best. As with the HOF approach, error generally decreases as population size increases. The best supporting evidence in these results is from window sizes 5 and 9. In most situations LOF outperformed HOF. Three of four exceptions to this occur for population 20. The other instance where HOF produced a smaller error is window size 9 with population 50. As with the HOF approach, all window and population size combinations perform better than the best baseline error. The above results confirm the conclusions drawn from the HOF approach and suggest that, at least for this function, the LOF approach performs better than the HOF approach.

4.2 Example 2

Polynomial: $3.8x^{16} - 4.2x^{14} - 2x^9 + 3.1x^2$

Function Domain: $[-1, 1]$ Parameter Range: $[-5, 5]$

4.2.1 HOF

Table 3	Population Size				
Win Size	20	50	100	150	Mean
1	52.98 3,5,7,9:-	45.00 3,5,7,9:20	43.49 3,5,7,9:20,50	43.27 3,5,7,9:20,50,100	46.19
3	154.64 5,7,9:-	91.61 5,7,9:20	71.76 5,7,9:20,50	63.23 5,7,9:20,50,100	95.31
5	243.29 7,9:-	145.62 7,9:20	113.07 7,9:20,50	99.62 7,9:20,50,100	150.40
7	326.94 9:-	174.53 9:20	117.38 9:20,50	106.41 -:20,50	181.36
9	445.42 :-	200.54 :-20	119.87 -:20,50	105.46 -:20,50,100	217.82
all	244.65 :-	131.46 :-20	93.11 1:20,50	83.60 1,5:20,50,100	138.21

Table 3 shows the average error for each population size and for the all-at-once GA approach. The results show that window size 1 performs statistically significantly better than all other window sizes for all populations. There is also a trend for error to statistically significantly increase as window size increases but there appears to be a maximum error which varies with both window and population size. These results support the earlier conclusions but, since window size 1 clearly outperforms all other window sizes, the optimum window size may be independent of population size but be dependent on the form of the function.

4.2.2 LOF

Table 4	Population Size				
Win Size	20	50	100	150	Mean
1	65.65 3,5,7,9:-	57.37 3,5,7,9:20	55.40 5,7,9:20,50	55.21 5,7:20,50	58.41
3	136.16 9:-	73.65 7,9:20	61.89 -:20,50	59.57 5:20,50	82.82
5	175.58 9:-	89.27 :-20	71.39 -:20,50	74.47 -:20,50	102.68
7	201.91 :-	109.79 :-20	76.83 -:20,50	65.26 -:20,50	113.45
9	245.43 :-	109.02 :-20	74.19 -:20,50	68.01 -:20,50	124.16
all	244.65 :-	131.46 :-20	93.11 1:20,50	83.60 1,5:20,50,100	138.21

Table 4 shows that window size 1 performs statistically significantly better than all other window sizes for populations 20 and 50. For population 100, window size 1 performs best, but not statistically significantly better

than window size 3. For population size 150, window size 1 is not statistically significantly better than window sizes 3 or 9. The same error to window size trend appears but in these results, however, these trends are not as statistically significant. The LOF approach performs worse than the HOF approach for window size 1 but better for all other window sizes.

5 Conclusions

It has been shown that searching all coefficients at once does not produce the best results for this problem domain. Both window size and population size have significant impact on error rates. The optimum window size may have some dependence on the population size but there is no consistent outcome in such a limited study. In all situations, larger population sizes perform better, as would be expected, but also have longer execution times. Populations of 20 took 20-30 minutes to run 100 trials, populations of 150 took approximately 2 hours. These longer execution times were part of the motivation for this work.

All experiments performed very well with window sizes of 3 or less and we suggest that larger window sizes will generally perform worse than smaller window sizes. It is surprising that the window size of 1 performed so well with polynomial 2 but a possible explanation for this concerns the amount of interaction between coefficients for different window sizes. When a window size of 1 is used, there is no interaction between terms. A window size 3 provides interaction between coefficients of neighbouring terms, but not distant terms. It is hypothesised that interactions between coefficients of neighbouring terms of the same parity (e.g. coefficients of x and x^3 , and x^3 and x^5) are advantageous. Polynomial 2 had no such neighbouring terms and was thus amenable to solution using a window size of 1. The results suggest that a window size of 3 is adequate for a good solution as well as good performance. Population sizes have a weak but monotonic influence on the optimum window size with larger populations requiring a larger window size for maximal accuracy. However, large populations still give good solutions with small window sizes. In most situations, the LOF approach performs better than the HOF approach. This is satisfying as it allows the process to be terminated when error ceases to decrease and does not need any arbitrary control as HOF would require. Although the objective function currently used appears to perform sufficiently well, it is unlikely to be useful for all functions.

In all situations the Window Algorithm performed better than the all-at-once basic GA. This gives a clear indication that the Window Algorithm has the potential to be a significant and useful modification to the standard GA.

References

- [Cardell et al., 1994] Cardell, N., Joerding, W., and Li, Y. (1994). Why some feedforward networks cannot learn some polynomials. *Neural Computation*, 6(4):761–766.
- [Caruna and Schaffer, 1988] Caruna, R. and Schaffer, J. (1988). Representation and hidden bias: Gray vs binary coding for genetic algorithms. In *Proceedings of 5th International Conference on Machine Learning*.
- [Ginsberg, 1993] Ginsberg, M. (1993). *Essentials of Artificial Intelligence*. Morgan Kaufmann.
- [Goldberg, 1989] Goldberg, D. (1989). *Genetic algorithms in Search, Optimization & Machine Learning*. Addison-Wesley.
- [Hecht-Nielsen, 1987] Hecht-Nielsen, R. (1987). *Neurocomputing*. Addison-Wesley.
- [Janikow and Michalewicz, 1991] Janikow, C. and Michalewicz, Z. (1991). An experimental comparison of binary and floating point representation in genetic algorithms. In *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 31–36.
- [Mathias and Whitley, 1994a] Mathias, K. and Whitley, D. (1994a). Changing representation during search: a comparative study of delta coding. *Evolutionary computation*, 2(3):249–278.
- [Mathias and Whitley, 1994b] Mathias, K. and Whitley, D. (1994b). Initial performance comparisons for the delta coding algorithm. In *IEEE Conf. on Evolutionary Computation*, volume 1, pages 433–438.
- [Michalewicz and Janikow, 1991] Michalewicz, Z. and Janikow, C. (1991). Handling constraints in genetic algorithms. In *Proceeding of 4th International Conference on Genetic Algorithms*.
- [Scheid, 1968] Scheid, F. (1968). *Theory and Problems of Numerical Analysis*. McGraw-Hill.
- [Srinivas and Patnaik, 1994] Srinivas, M. and Patnaik, L. (1994). Genetic algorithms: a survey. *Computer*, 27:17–26.
- [Whitley et al., 1991] Whitley, D., Mathias, K., and Fitzhorn, P. (1991). Delta coding: an iterative search strategy for genetic algorithms. In *Proceedings of 4th International Conference on genetic algorithms*.
- [Wright, 1993] Wright, A. (1993). Genetic algorithms for real parameter optimisation. In Rawlins, G., editor, *Foundations of genetic algorithms*. Morgan Kaufmann.