Time Complexity Estimation and Optimisation of the Genetic Algorithm Clustering Method

Z. M. NOPIAH, M. I. KHAIRIR, S. ABDULLAH, M. N. BAHARIN, AND A. ARIFIN
Department of Mechanical and Materials Engineering
Universiti Kebangsaan Malaysia
43600 UKM Bangi, Selangor
MALAYSIA
zmn@eng.ukm.my

Abstract: - This paper presents the time complexity estimation and optimisation of the genetic algorithm clustering method. The tested feature in the clustering algorithm is the population limit function. For the purpose of the study, segmental kurtosis analysis was done on several segmented fatigue time series data, which are then represented in two-dimensional heteroscaled datasets. These datasets are then clustered using the genetic algorithm clustering method and the runtime of the algorithm is measured against the number of iterations. Polynomial fitting is used on the runtime data to determine the time complexity of the algorithm. Analysis is repeated with the inclusion of the population limit in the clustering algorithm. The results of the analysis will be used to determine the significance of including the population limit function in the algorithm for optimal performance.

Key-Words: - Genetic algorithms, fatigue damage, clustering, time complexity, big-O notation, algorithm efficiency.

1 Introduction

In the field of evolutionary computing, the evolutionary principles of survival of the fittest, natural selection and genetic inheritance are abstracted and modeled into algorithms that search for optimal solutions to a problem. The most popular technique in evolutionary computing research has been the genetic algorithm [1-3]. Genetic algorithms (GA) perform meta-heuristic search in complex, large, and multimodal landscapes, and provide near-optimal solutions for objective or fitness functions of optimization problems[4-5]. GAs and GA-based techniques have been used in fields such as industrial engineering [1], clustering [6-9] and in optimizing the performance of neural networks, fuzzy systems, production systems, wireless systems and other program structures [2].

A GA-based clustering method was developed for applications in cluster analysis of heteroscaled datasets. Since this method is to be applied numerous times across multiple datasets in numerous iterations, it is necessary that the method performs efficiently in real time in order to consistently produce efficient results without using up too much computing and time resources. This study aims to analyze the time complexity of the GA clustering algorithm by comparing the real time performance of the method with and without a population limit function. The results will be used to determine whether or not the population limit function must be included in the algorithm for optimal performance.

2 Literature Review

2.1 Genetic Algorithms

Most GA methods have at least the following in common: populations of chromosomes, selection according to fitness, crossover to produce new offspring, and random mutation of new offspring [2]. Solutions in GA are encoded as chromosomes which are strings of numbers or characters that represent the values or parameters of the solution to the problem. The chromosomes are commonly encoded as strings of binary, real-valued, integer, octal, or hexadecimal numbers [1,10-11]. Each of these types of number have their own advantages and disadvantages when used for certain data types or for searching for solutions to certain problems. In this study, real-valued number strings were selected as the chromosome encoding for the population of potential solutions.

The set of potential solutions to the problem is represented as a population of chromosomes. Initially, a random population is created, which represents different points in the search space of potential solutions [4,12-13]. A fitness function assigns a score (fitness) to each chromosome in the current population, which will determine its survival into the next generation. The fitness of a chromosome depends on how well that chromosome can solve the problem at hand [12].

The selection of chromosomes is done on the current population based on the fitness values – chromosomes with higher fitness are more likely to be selected than those with low fitness values. This is mostly done using probabilistic methods; in evolutionary computing research, the common methods of selection are the roulette wheel, tournament, and rank selection [1-2, 4]. Selected chromosomes are then included in the next generation of population.

Next the population undergoes the crossover (also called recombination) genetic operator, which selects chromosomes from the population to produce offsprings. Using random selection or any of the previously mentioned selection methods, two parent chromosomes are chosen for crossover operation. Using single-point [7], two-point, or *N*-point crossover, parts of the gene string in each parent chromosome are swapped to produce two new offspring, which are included in the next generation of population. The process is repeated a number of times, usually according to some user-specified proportional value of the current population [14].

Random chromosomes from the surviving population are selected for mutation, where some random part of the chromosome's gene is arbitrarily changed. This genetic operation may or may not yield superior offspring, but it ensures that solutions are not trapped in local extrema. Mutation is performed according to some degree of probability [14], usually very small, so that the GA does not approximate a random search [1].

The process of selection, crossover, and mutation are then repeated on the surviving population, until some terminating criteria is reached, i.e. a maximum number of generations, a minimum change in population fitness, etc. The resulting final population is then considered to be the set of solutions that best solves the problem at hand. The best individual chromosome (the chromosome with the highest fitness value) in the final population is usually determined to be the optimal solution to the problem.

2.2 GA-based Clustering

Several MATLAB functions were written and developed for the purpose of having a dynamic and robust algorithm that can be used for a large variety of two-dimensional datasets. The methods employed in this algorithm were partly derived from a GA-based clustering technique developed in 2000 by Maulik and Bandyopadhyay [5].

Each chromosome of the population was encoded as the horizontal concatenation, i.e. [[1 2] [3 4] …], of location vectors of the centroids. For uniformity we set the number of clusters *k* to be equal to 3 for all datasets. Each of the 3 centroids' location is determined by a pair of vectors, therefore the length of each chromosome in the population is 6, where each of the gene location pairs [1 2], [3 4], and [5 6] denote the location vectors of the cluster centroids in two-dimensional space.

The initial population of chromosomes was randomly generated within the ranges of both dimensions of the dataset. The initial population size *n* was set to be 20 and subsequent generations may or may not have the same population size. The fitness of each individual chromosome in the population was then calculated as follows:

Step 1: Normalize both dimensions of the data by dividing the values of the data by the maximum value of the data on each dimension.

Step 2: Calculate the distance d_j of each datapoint from each centroid using the square Euclidean distance formula

$$\{d_j\}_{i,r} = (x_i - p_{j,r})^2 + (y_i - q_{j,r})^2 \tag{1}$$

where $i = 1, 2, \ldots, m$, $j = 1, 2, 3$ and $r = 1, 2, \ldots, n$; (x_i, y_i) is the location of the *i*th datapoint in a dataset of *m* points and $(p_{j,r}, q_{j,r})$ is the location of the *j*th centroid in the *r*th chromosome of the current population.

Step 3: Determine the smallest distance value in the set of distances $\{ d_1, d_2, d_3 \}_{i,r}$ for each of the *m* datapoints and each of the *n* chromosomes.

Step 4: Sum up all the smallest distance values across all the *m* datapoints, resulting in *n* values of summation. The reciprocal of each summation value is the fitness value f_i for its corresponding chromosome.

The normalization of all dimensions of the data in ***Step 1*** was incorporated into the fitness evaluation procedure to minimize or nullify the effect of the differently scaled dimensions of the dataset on the clustering of data.

The population selection method used was a modified fitness proportionate (roulette wheel) selection technique. The probability of the *i*th chromosome to be selected is defined as

$$p_i = \frac{f_i}{\sum_{j=1}^{n} f_j} \tag{2}$$

In Equation 2 the integer *n* is the size of the current population of chromosomes. Next, a random number in the range (0, 1) was chosen and multiplied with the maximum value of probability computed, and if this number is smaller than p_i then that chromosome is selected into the next population, otherwise the chromosome doesn't survive into the next generation. Using this method, chromosomes with higher fitness values have a better chance of being selected into the next population, and chromosomes with smaller fitness values have a better chance of dying out and not included in the next population of solutions.

Next, the crossover operation was performed where a pair of chromosomes was randomly selected from the current population. Then two integer values in the range [1, 6] were randomly chosen to represent the crossover points, where the gene values between the two points inclusive were swapped between the parent chromosomes, producing two offsprings. If the two integers were distinct, the operation was a two-point crossover and if the two integers were equal, the operation was a single-point crossover. The crossover operation was then repeated on the population until some user-determined proportion of the population (crossover rate) had been selected for breeding.

Finally the resulting population was then run under the mutation operator, where a chromosome was randomly chosen for mutation. Then a random gene location of the chromosome was chosen and its value v changed into a mutated gene value v' according to the following operation:

$$v' = v \pm (a + b) * v \tag{3}$$

In Equation 3, a and b are random values in the range (0, 1). These values are important in order to minimize the chances that the population be trapped in a local minimum, which is the main objective of the mutation operator. The mutation operation was also repeated on the population until some user-determined proportion of the population (mutation rate) had been selected for mutation.

```
function vargout = gacluster(2Ddata, vargin)

initPop = gen_pop(2Ddata, num_clustr);

while count <= max_repetition

  PopFit = fitness_eval(initPop);
  nextPop = select(initPop, PopFit);
  nextPop = crossover(nextPop, rc);
  nextPop = mutate(nextPop, mc);

  initPop = nextPop;
  nextPop = [];

end

FinalPop = initPop;

best_value = bestfit_ind(FinalPop);

vargout = cluster(2Ddata, best_value)
```

Fig. 1: The pseudocode for the GA clustering method

The resulting population is labeled as the next generation of solutions, and the whole algorithm is repeated on this generation and each successive generation until it meets the termination criterion, which is the maximum number of generations, in this case set at 1000. The best individual chromosome from this final population is then chosen to be the location vector of the cluster centroids for the particular dataset.

2.3 Time complexity analysis

Time complexity analysis is a part of computational complexity theory that is used to describe an algorithm's use of computational resources; in this case, the worst case running time expressed as a function of its input using big Omicron (big-O) notation [15-16,18]. The big-O notation is used to express the upper bound of the growth rate of a function and is mostly used to describe asymptotic behavior [15,17] or optimal rate of convergence [20].

The big-O notation is described using set notation as follows:

$$O(g(n)) = \{f \mid \exists c>0, \exists n_0 > 0, \forall n \geq n_0 : 0 \leq f \leq cg(n)\} \tag{4}$$

In other words, $f \in (g(n))$ if and only if there exist positive constants c and n_0 such that for all $n \geq n_0$, the inequality $0 \leq f \leq cg(n)$ is satisfied. We say that f is big O of $g(n)$, or that $g(n)$ is an asymptotic upper bound for f [9].

In terms of time complexity analysis, we use the term $T(n) \in O(g(n))$ and say that the algorithm has order of $g(n)$ complexity. This means that the time taken to compute a problem of size n is in the set of functions described by $O(g(n))$.

Time complexity analysis can be used to predict the growth behavior of an algorithm and is useful for analyzing and optimizing the real time efficiency of the algorithm [16] or performing a worst case analysis of a computation process [19].

3 Methodology

In this study, several segmented fatigue time series data (see Table 1 and Figure 2) that were segmented using the previously developed Peak-Valley segmentation method [21-23] were used to test the GA clustering algorithm for real time efficiency. Segmental kurtosis analysis was done on each segmented fatigue data, and the results are represented in two-dimensional heteroscaled datasets.

The GA clustering algorithm is then used on these datasets to cluster fatigue damage segments based on their kurtosis and fatigue damage values. Simultaneously, the number solutions in the population and the running time are recorded while the algorithm is running. For the purpose of this study, the algorithm is set to run until 1000 generations have been produced, which means that the algorithm has iterated 1000 times. The recorded runtime and population growth are then plotted and polynomial fitting is used to estimate the growth function of the running time.

Table 1: Description of datasets used in the study

Dataset	Description
SAETRN	SAE transmission test fatigue data
DK1	Pavé road loading on lower arm suspension
DDK2	Highway road loading on lower arm suspension

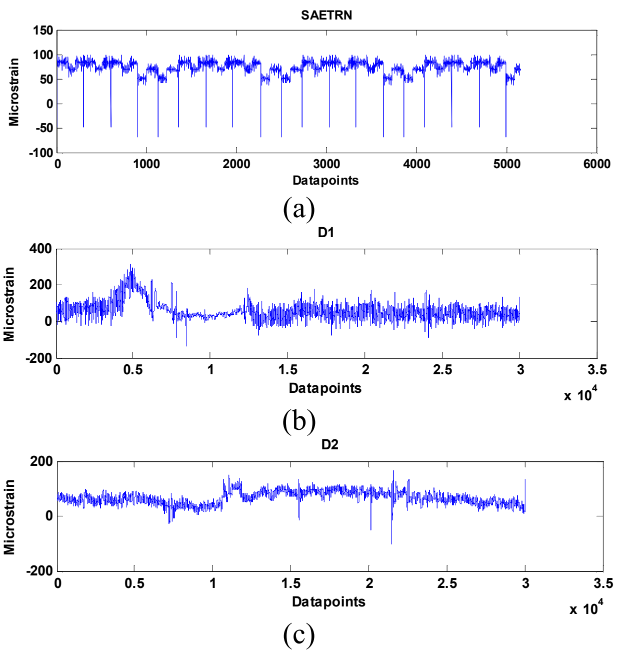


Fig. 2: Fatigue time series data for (a) SAETRN, (b) D1, and (c) D2

The processes above are then repeated on the heteroscaled datasets after a population limit function is included in the selection process of the GA clustering algorithm. The theoretical value of the population limit of the GA clustering algorithm is evaluated as

$$P_{max} = C_P P_0 (1 + r_c)(1 + r_m) \tag{5}$$

Equation 5 is derived from how the population size grows with every iteration of the GA clustering algorithm. Initially, the population size is a positive integer P_0. For a worst case scenario, we assume that the whole population was selected in the selection process. This means that P_0 number of solutions is considered for the crossover operation.

A portion of the population is selected for the crossover process, which produces additional solutions to be added to the population. Let r_c be the crossover rate where $0 \le r_c \le 1$; the number of additional solutions would be $P_0 r_c$, making the total number of solutions in the population so far to be $P_0 + P_0 r_c$.

Next, a portion of this population is selected for the mutation process. Let r_m be the value of the mutation rate, where $0 \le r_m \le 1$; the number of solutions added to the population is $(P_0 + P_0 r_c)\, r_m$; therefore the total number of solutions in the population after the mutation process is $P_0 + P_0 r_c + (P_0 + P_0 r_c)\, r_m$.

The expression above is then simplified using factorization as follows:

$$\begin{aligned} &P_0 + P_0 r_c + (P_0 + P_0 r_c)\, r_m \\ &\quad = P_0 (1 + r_c + (1 + r_c)\, r_m) \\ &\quad = P_0 (1 + r_c)(1 + r_m) \end{aligned} \tag{6}$$

In order to enable the user to have some amount of control over the maximum population, the expression above is multiplied with a user-defined coefficient of population C_P, which is a positive real number. This results in Equation 6 as expressed above. For the purpose of this study, the values P_0, C_P, r_c and r_m are set to be 20, 2, 0.4 and 0.1 respectively.

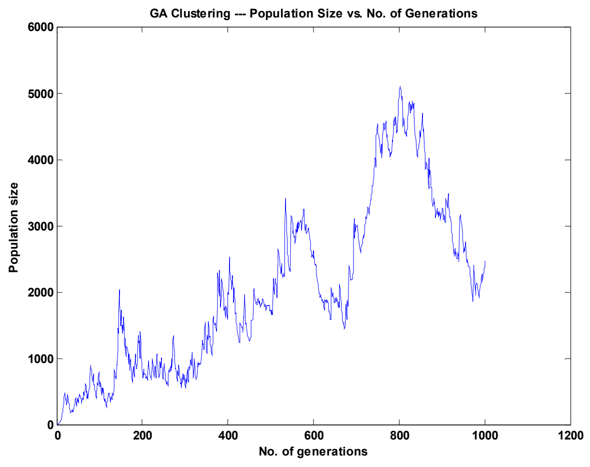
As with the previous set of data, the population size and running time of the algorithm is recorded simultaneously as the population limited GA clustering algorithm is run on the datasets. The recorded runtime and population growth are then plotted and polynomial fitting is used to estimate the growth function of the running time.

The time complexity of the algorithm is then determined from the fitted growth function. The results are then compared for the GA clustered and the population limited GA clustered datasets to determine the significance of including the population limit function in the GA clustering algorithm.

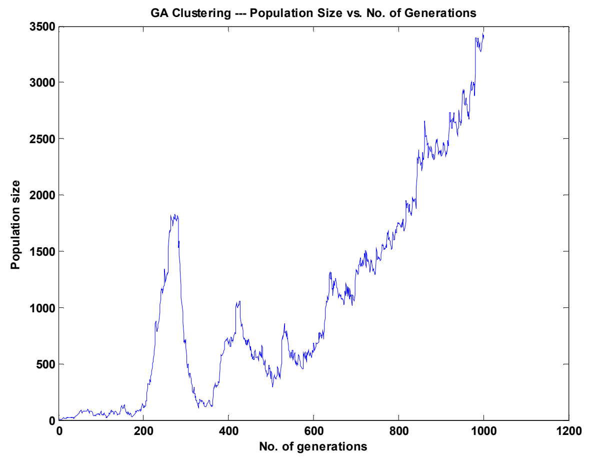
4 Results and Discussion

For the purpose of understanding the population growth and the time complexity of the GA clustering algorithm, plots of population growth and polynomial fitted runtime are observed and compared.

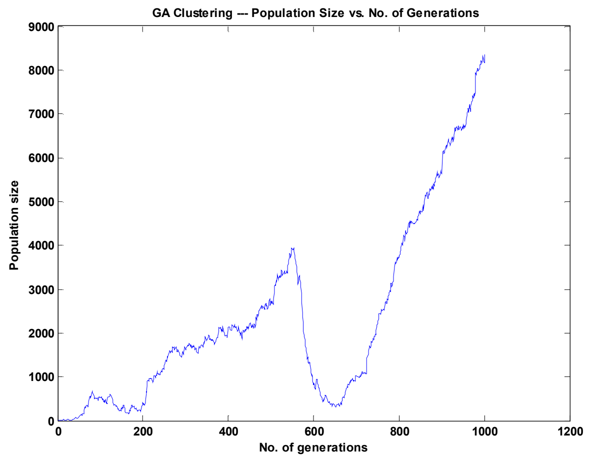
Figure 3 shows how the population size grows with the number of generations or iterations of the GA clustering algorithm. We can see that it is apparent that for all datasets, although the population sizes increase and decrease erratically with the number of iterations, overall they generally exhibit a positive growth behavior. This means that eventually, after some large enough number of generations, the population size continually increases as the number of generation increases. The increase of the size of the population will in turn increase the problem size for the GA clustering algorithm, which will affect the time complexity of the algorithm. Generally, a larger problem size means larger computing time or resource is needed for the algorithm to complete its task.



(a)

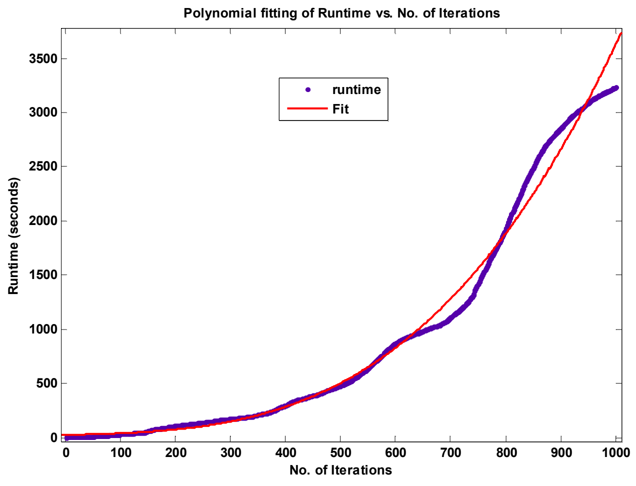


(b)

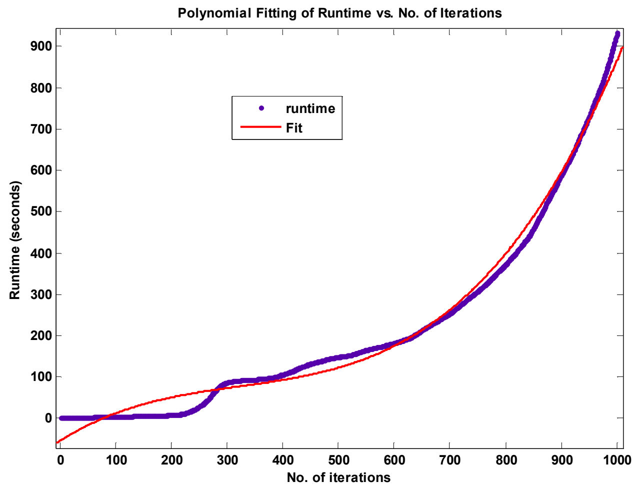


(c)

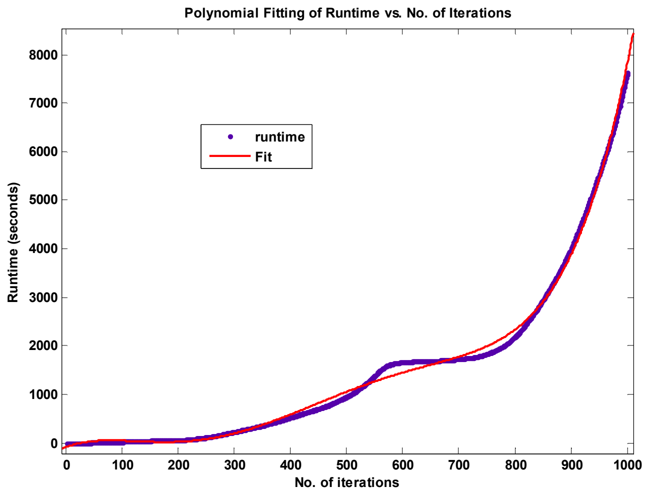
Fig. 3: Population size versus number of generations for (a) SAETRN, (b) DK1, and (c) DDK2



(a)

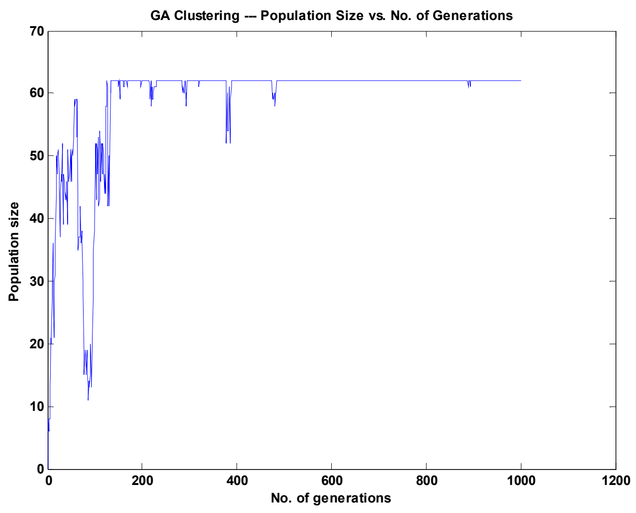


(b)

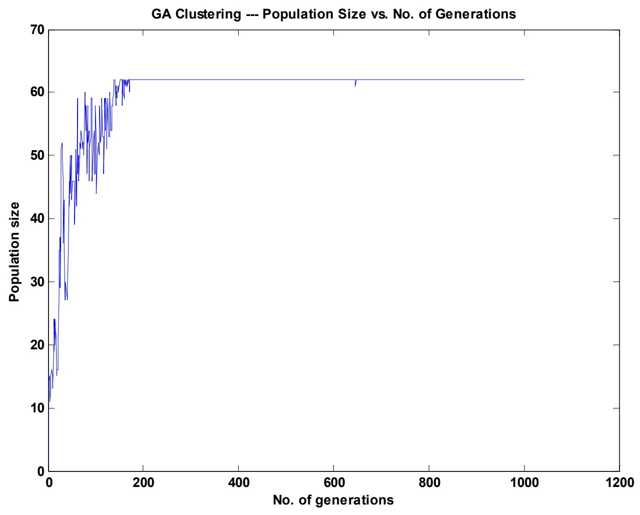


(c)

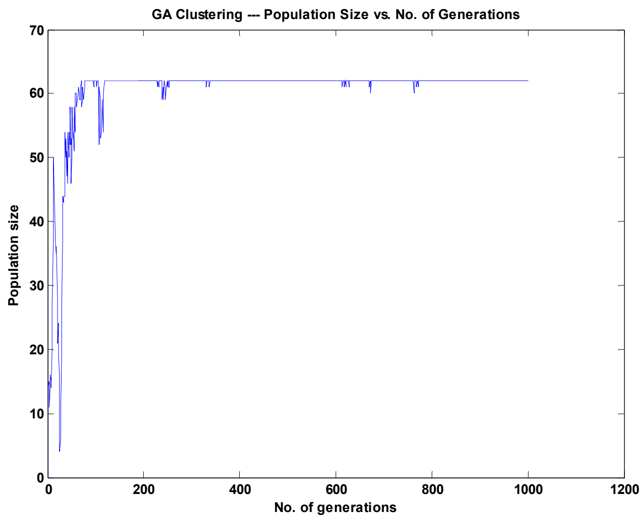
Fig. 4: Polynomial fitting of runtime versus number of iterations for (a) SAETRN, (b) DK1, and (c) DDK2



(a)

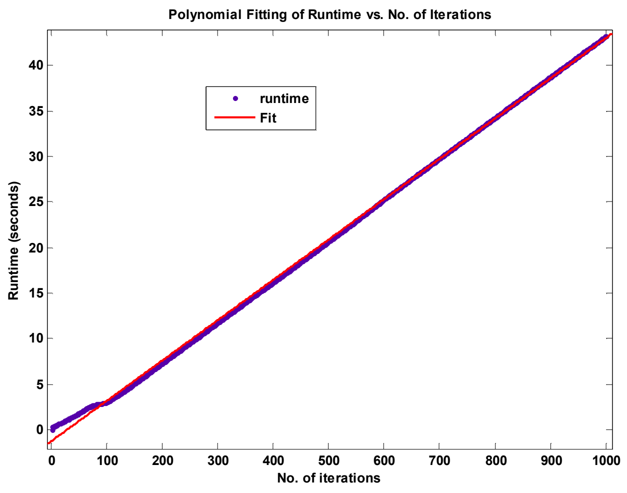


(b)

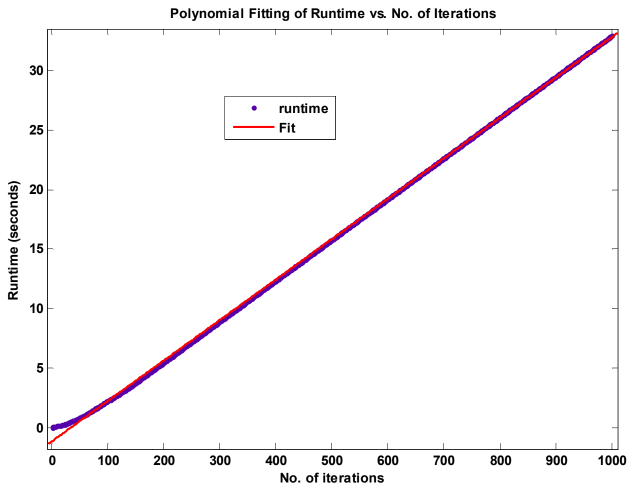


(c)

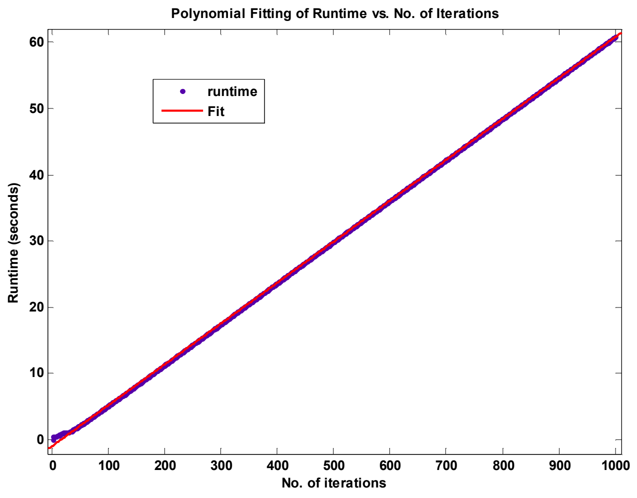
Fig. 5: Population size versus number of generations for (a) SAETRN, (b) DK1, and (c) DDK2 when population limit function is applied



(a)



(b)



(c)

Fig. 6: Polynomial fitting of runtime versus number of iterations for (a) SAETRN, (b) DK1, and (c) DDK2 when population limit function is applied

Table 2: Fitted models and complexity

Dataset	Fitted model	Goodness of fit (R^2)	Complexity
SAETRN	$f(x) = p_1x^3 + p_2x^2 + p_3x + p_4$	0.9895	$O(n^3)$
DK1	$f(x) = p_1x^3 + p_2x^2 + p_3x + p_4$	0.9897	$O(n^3)$
DDK2	$f(x) = p_1x^5 + p_2x^4 + p_3x^3 + p_4x^2 + p_5x + p_6$	0.9967	$O(n^5)$

Table 3: Comparison between non-limited (N) and limited (L) population and runtime

Data	Population			Runtime (seconds)		
	N	L	%Diff	N	L	%Diff
SAETRN	5086	62	98.8	3236	43	98.7
DK1	3432	62	98.2	932	33	96.5
DDK2	8352	62	99.3	7624	61	99.2

Figure 4 shows the actual running time (in seconds) versus the number of iterations and the fitted model used to predict the asymptotic behavior of the runtime for each dataset. We can see that the algorithm runs in polynomial time of some degree, and the fitted models estimate the order of polynomial time the algorithm runs in for each dataset. Table 2 shows the fitted models for the runtime for each dataset and the estimated orders of polynomial time. For datasets SAETRN and DK1, the time complexity is $O(n^3)$ and for DDK2 the time complexity is $O(n^5)$. This means that for two datasets, the algorithm runs in cubic time and for the other dataset, the algorithm runs in polynomial time of degree 5. This tells us that since the population size grows unboundedly, the problem size also grows unboundedly and therefore a much larger computing resource is needed for each next iteration of the GA clustering algorithm. Consequently, the algorithm's running time also increases in the order of polynomial degree 3 or 5 depending on the data. These polynomial growth rates are undesirable for optimum algorithm efficiency, since larger problem sizes would require significantly longer runtime periods and much larger computing resources.

Figure 5 shows the population growth when the GA clustering algorithm is modified to include the population limit function in its selection process. We can see that although the population sizes generally increase with the number of generations, the numbers are capped at a particular value P_{max} which can be obtained using Equation 5.

Figure 6 shows the actual running time in seconds versus the number of iterations and the fitted model used to predict the asymptotic behavior of the runtime for each dataset when the GA clustering algorithm is run with the population limit function included in its selection process. Based on the fitted models, it is clear that for all datasets, the time complexity of the algorithm is $O(n)$, which means that the algorithm runs in linear time. This tells us that the running time increases linearly with the number of iterations, and that each iteration requires some constant time to perform. This case is much more desirable over the case where the algorithm runs in polynomial time of degrees larger than 1. Since the population size is capped at some finite value P_{max}, the problem size ceases to grow unboundedly and consequently the computing resource needed for each iteration of the GA clustering algorithm is eventually capped, making the computing time for each iteration constant.

Improvements in both the population size and runtime is shown in Table 3. We can see that for all datasets, limiting the population results in 98.2% to 99.3% reduction in population, which in turn reduces the problem size drastically. This is also clearly reflected in the improvements in algorithm running times. For all datasets, limiting the population reduces the running time by 96.5% up to 99.2%. What usually took up to two hours to execute now takes only one minute to produce results if the population limit function is incorporated in the selection process. This clearly shows the advantages of capping the problem size and consequently reducing running time of the algorithm, which saves both computing time and resources.

The results of the time complexity analysis shows that the population limit function in the selection process of the GA clustering algorithm has managed to reduce the time complexity of the algorithm from higher degree polynomial time to linear time. This means that by including the population limit function in the GA clustering algorithm, the running time of the algorithm can be significantly reduced and the user will have some degree of control over the complexity of the algorithm in both computing time and resource.

5 Conclusion

Performing time complexity analysis on the GA clustering algorithm has helped us to determine how the algorithm performs in real time as the problem size increases. It has been found that including the population limit function in the selection process of the GA clustering algorithm will reduce the time complexity of the algorithm to linear time. This significant reduction in time complexity will be very useful in future developments of the GA clustering algorithm, particularly for clustering larger datasets in higher dimensions.

6 Acknowledgements

The authors would like to express their gratitude to Universiti Kebangsaan Malaysia and Ministry of Higher Education, through the fund of UKM-GUP-BTT-07-25-152, for supporting these research activities.

References:

[1] S.N. Sivanandam, and S.N. Deepa, *Introduction to Genetic Algorithms*, New York: Springer-Verlag Berlin Heidelberg, 2008.

[2] M. Gen and R. Cheng, *Genetic Algorithms and Engineering Optimization*, New York: Wiley-Interscience, 2000.

[3] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*, 3rd rev. ext. ed. New York: Springer-Verlag, 1996.

[4] M. Mitchell, *An Introduction to Genetic Algorithms*, 5th print, Cambridge, MA: The MIT Press, 1999.

[5] U. Maulik and S. Bandyopadhyay, Genetic algorithm-based clustering technique. *Pattern Recognition*, Vol. 33, 2000, pp. 1455-1465.

[6] W. Song, C.H. Li and S.C. Park, Genetic algorithm for text clustering using ontology and evaluating the validity of various semantic similarity measures. *Expert Systems with Applications*, Vol. 36, 2009, pp. 9095-9104.

[7] J.-L. Lin and M.-C. Wei, Genetic algorithm-based clustering approach for k-anonymization. *Expert Systems with Applications*, 2009, doi: 10.1016/j.eswa.2009.02.009

[8] Z. M. Nopiah, M. I. Khairir, S. Abdullah and M. N. Baharin, A weighted genetic algorithm based method for clustering of heteroscaled datasets. *Proceedings of the 2009 International Conference on Signal Processing Systems (ICSPS 2009)*, Singapore, 15-17 May 2009, pp. 971-975.

[9] Z. M. Nopiah, M. I. Khairir, S. Abdullah and M. N. Baharin, A genetic algorithm based method for normalized clustering of heteroscaled datasets. *Proceedings of the 5th International Conference on Mathematics, Statistics and Their Applications (ICMSA 2009)*, Bukittinggi, Indonesia, 9-11 June 2009.

[10] L.A.N. Lorena and J.C. Furtado, Constructive genetic algorithm for clustering problems. *Evolutionary Computation*, Vol. 9, No. 3, 2001.

[11] M.-H. Wang, Y.-F. Tseng, H.-C. Chen and K.-H. Chao, A novel clustering algorithm based on the extension theory and genetic algorithm. *Expert Systems with Applications*, Vol. 36, 2009, pp. 8269-8276.

[12] N.H. Park, C.W. Ahn and R.S. Ramakrishna, Adaptive clustering technique using genetic algorithms. *IEICE Transactions on Information and Systems*, Vol. E88-D (12), 2005, pp. 2880-2882.

[13] A. Banerjee and S.J. Louis, A recursive clustering methodology using a genetic algorithm. *IEEE Congress on Evolutionary Computation (CEC)*, 2007, pp. 2165-2172.

[14] J. A. Hageman, R. A. van den Berg, J. A. Westerhuis, M. J. van der Werf and A. K. Smilde, Genetic algorithm based two-mode clustering of metabolomics data. *Metabolomics*, Vol. 4, 2009, pp. 141-149.

[15] Knuth, D. E., Big Omicron and big Omega and big Theta. *SIGACT News* Vol. 8, No. 2 (Apr. 1976), 1976, pp. 18-24.

[16] Black, P. E., big-O notation, *Dictionary of Algorithms and Data Structures* [online], U.S. National Institute of Standards and Technology, 2008 (accessed 26 November 2009). Available from: http://www.itl.nist.gov/div897/sqg/dads/HTML/bigOnotation.html

[17] N. Chen and Z. Yan, Complexity analysis of Reed-Solomon decoding over GF(2^m) without using syndromes. *EURASIP Journal on Wireless Communications and Networking*, 2008, Article ID 843634, 11 pages, 2008. doi:10.1155/2008/843634

[18] H.S. He, Forest landscape models: definitions, characterization, and classification. *Forest Ecology*

and Management, Vol. 254, No. 3, 2008, pp. 484-498.

[19] B.J. Waterhouse, F.Y. Kuo and I.H. Sloan, Randomly shifted lattice rules on the unit cube for unbounded integrands in high dimensions, *Journal of Complexity*, Vol. 22, No. 1, 2006, pp. 71-101.

[20] F.Y. Kuo, Component-by-component constructions achieve the optimal rate of convergence for multivariate integration in weighted Korobov and Sobolev spaces, *Journal of Complexity*, Vol. 19, No. 3, 2003, pp. 301-320.

[21] Z.M. Nopiah, M.I. Khairir, S. Abdullah, & C.K.E. Nizwan. 2008. Peak-valley segmentation algorithm for fatigue time series data, *WSEAS Transactions on Mathematics*, Issue 12, Vol. 7, December 2008, pp. 698-707.

[22] Z.M. Nopiah, M.I. Khairir, S.Abdullah, M.N. Baharin & C.K.E. Nizwan. 2008. Peak-valley segmentation algorithm for fatigue time series data. *Proceedings of the 7th WSEAS International Conference on Computational Intelligence, Man-Machine Systems And Cybernetics (CIMMACS '08)*, *Cairo, Egypt, 27-31 Dec 2008*.

[23] Z.M. Nopiah, M.I. Khairir & S. Abdullah. 2008. Segmentation and scattering of fatigue time series data by kurtosis and root mean square. *Proceedings of the WSEAS International Conferences on Signal Processing, Istanbul, Turkey, May. 27-30, 2008*, pp. 64-68.