STUDY ON GA BASED SOLUTIONS TO SET PARTITIONING PROBLEM AND PROPOSED NEW APPROACH

S.M. Shahriar Nirjon
Department of CSE
Bangladesh University of Engineering & Technology
E-mail: nirjon_buet@yahoo.com

Abstract*—Evolutionary Algorithms (EAs) are search methods that take their inspiration from natural selection and survival of the fittest in the biological world. Genetic Algorithm is a kind of EA GAs emphasize on the behavioral linkage between the parent generation and their offspring. GAs have been successfully applied to many optimization problems in recent years. In this paper, we work with various GA based solutions used to partition a number of objects into a numbers of disjoint subsets according to some problem specific optimization criteria and we also pose the merits and demerits of these solutions. Later on we propose our own structural approach and show the results.*

Keywords*—Crossover, Fitness function, Genetic Algorithm, Mutation, Population, Selection.*

1. Introduction

Genetic Algorithm is a variant of stochastic beam search in which successor states are generated by combining two parent states rather than modifying a single state. The analogy to natural selection is the same as in stochastic beam search, except new states are generated by the process of reproduction. Offspring bears the genetic information present in the parent.

John Holland [3] (1975) is credited with inventing the area of Genetic Algorithm. According to his Schema Theorem, we can expect a population of individuals to evolve towards containing good solution candidates.

Partitioning a set of objects into a number of disjoint subsets is a well-known problem. It comes in various forms. For example, the Equal Piles Problem was defined and first studied by Jones and Beltramo (1991).The problem is to partition N numbers into K subsets, such that the sums of the subsets are as nearly equal as possible. (Jones and Beltramo relate the problem to the case of objects of specified heights, which are to be, stacked into disjoint piles so that the pile heights are as nearly equal as possible.) The optimization criterion here is to minimize the sum of differences from average.

At first, we discuss previous attempts on solving this problem with GAs. The fitness function, selection strategy, crossover and mutation operation of these approaches are described. Finally we present our approach and a summarized result of applying the algorithm over randomly generated data.

2. The Problem

There are N- objects, each associated with a value $X(i)$, $i = 1$ to N. We have to divide these N objects into K- nonempty disjoint subsets $S(j)$, $j = 1$ to K such that,

I) $\sum | S(j) | = N$, for $j = 1$ to K.

II) $| S(j) | > 0$, for $j = 1$ to K.

III) for all $i = 1$ to N, $X(i)$ belongs to some $S(j)$, $j = 1$ to K.

Let, $E(i, j)$ = the j-th element of i-th subset.

$$C(i) = \sum E(i, j) , j = 1 \text{ to } | S(i) | \quad (1)$$

$$Cn = \sum C(i), i = 1 \text{ to } K. \quad (2)$$

$$C_{av} = C_n / K \quad (3)$$

$$D(i) = | C(i) - C_{av} | \quad (4)$$

$$D = \sum D(i), i = 1 \text{ to } K \quad (5)$$

And our goal is to minimize D.

3. GA-Based Solutions' Literature Survey

3.1. Basic Stages

GA-based algorithms iterate over the basic stages: Fitness evaluation, Selection, Crossover, and Mutation. In the following sub-sections we describe some successful attempts so far by various researchers.

3. 1.1. Initial Population

Initial population is a set of vectors, $P_0 = \{p_0(1), p_0(2) \ldots p_0(k)\}$, where population size k -is a user defined parameter. A larger k-value will definitely exhaust large amount of memory and will also increase the computation time. On the other hand, a smaller k-value means smaller population and

therefore poor performance. Chu and Beasley [4] showed by their computation that the quality of the solution is not very sensitive to the size of the population, and a population of size k = 100 were used by them. They also generated the initial population randomly. Each of the vectors has N components. Let, $p_0(i, j)$ denote the j-th component of i-th vector $p_0(i)$ from initial population P_0. $p_0(i, j)$ is initialized according to uniform distribution over the range $[1, K]$. (The meaning of N, K is given in section 2). On the other hand, Greene [5] kept individual partitions in a population arranged in increasing order of error. Although this introduced more computation and complex data structure to manipulate individual sets, it produces good results

3.1.2. Fitness Evaluation

There are two approaches to defining the fitness of an individual. The most common approach is to use penalty function [6] [7]. Penalty methods allow constraint to be violated [4]. An alternative approach for defining fitness involves separating the single fitness measure into two values: one is called fitness value and other is unfitness value [4]. The limitations of the former approach are discussed in [6]. As described in section 2, our aim is to minimize D. Each of the k-vectors (i.e. k-individuals) in i-th generation P_i has corresponding error value $f_i(j), j = 1 \text{ to } k$. This $f_i(j)$ is the D –value for the j-th individual of i-th generation, which is calculated from equations (1), (2), (3), (4) and (5). The overall error value of apopulation F_i, is defined as the minimum of all $f_i(j)$. i.e. $F_i = \min \{ f_i(1), f_i(2) \ldots f_i(k) \}$. In Greene [5], the fitness is just the complement of error; the more erroneous a partition, the less fit it is. It's the simplest of all approaches.

3.1.3. Parent Selection

Roulette-wheel selection scheme (also known as stochastic sampling with replacement) is a popular method for selection. In this method individuals are mapped to contiguous segments of a line, such that each individual's segment is proportional to its fitness. A random number is generated and the individual whose segment spans the random number is selected.

Survival of the fittest surfaces in two forms [5]. Firstly, *elitism* is practiced: a (small) percentage of the best individuals automatically survive into the next generation. Secondly, a weighted roulette wheel (Goldberg 1989) is used to favor fitter (less erroneous) parents as candidates for mating with crossover.

Since we are minimizing $f_i(j)$, we have to map this $f_i(j)$ values to another value $J_i(j) = (\sum f_i(m), m = 1 \text{ to } k) - f_i(j)$. Now this individual with smaller $f_i(j)$ will have larger probability of being selected for crossover.

Chu and Beasley [4] developed Maximum Compatibility Selection (MCS) method for selecting parents that attempts to improve solution quality as well as feasibility. Their approach divides the fitness unfitness landscape into 4 subgroups. One parent is selected using a binary tournament based on fitness and then the other one is selected to give a maximum compatibility score.

3.1.4. Crossover

Single or multi-point crossover method is often applied to generate new population. Two of the selected individuals are taken and some crossover point is selected uniformly at random within the range $[1, N]$. The variables are exchanged between the individuals about this point, and then two new offspring are produced.

Jones and Beltramo [8] tried nine GAs, the best of which turned out to be an ordering GA with PMX crossover (Goldberg 1989). But on an average the result was not satisfactory.

Falkenauer [9] identified the problem and proposed that the entire subset should be treated as a gene rather than individual numbers. Falkenauer applied his Grouping Genetic Algorithm(GGA) on this problem and it showed a better result. Greene [5] proposed an interesting approach that he called 'Eager Breeder' algorithm. Here k best subsets are at first selected from a total of 2k subsets (k subsets from each parent). It starts two pointers, one at the beginning of each parental subset list, and let them track down their respective lists. At each step, the better subset of the pair now pointed at is copied into the child's collection. Then an adjustment is done when an object belongs to more than one subset or no subset using most into least (MIL) heuristic. Falkenauer employs this same insight and terms it the Equal Piles Descending (EPD) heuristic.

3.1.5. Mutation

Offspring are mutated after being created by recombination with a low probability. Our Mutation probability follows exponential distribution. The more the value of $J_i(j), j = 1 \text{ to } k$ the less is has probability of being mutated. Our mutation probability is: $\lambda \exp(-\lambda Ji(j))$, where λ is a predefined parameter. As mentioned earlier, in Greene [5] the population is kept sorted according

to the fitness value. Then P_t is divided into four bands. The first band size equals the size of elite survivors (10%) and goes through just one mutation step. The remaining 3 bands consist of the next 33%, 30% and 30% of the population in decreasing fitness. With 50% probability an element of these bands undergoes 4, 10, 20 mutation steps. Mutation operator designed by Chu and Beasley [4] is of two types – static and dynamic. Mutation rate (Ms) is constant at static mutation. Dynamic mutation is helpful when an attempt to satisfy a particular constraint may introduce infeasibilities into other current feasible constraints. Whenever certain constraint is satisfied in less than a fraction of the total population *(i.e. £ | P |, £ small)*, dynamic mutation tries to re-introduce solutions that satisfies that constraint.

3.2. Framework of GA based Algorithms

Following is the common framework of any GA based algorithm:

Algorithm GA_Search

1. Generate an initial population;
2. Evaluate fitness of individuals in the population;
3. **repeat**
4. Select parents from the population;
5. Recombine (mate) parents;
6. Evaluate fitness of the children;
7. Mutation;
8. **until** a satisfactory solution is found;

4. Proposed Algorithm for Set Partitioning Problem

We now propose a different structure rather than the conventional one:

Algorithm GA_Caller (α , β) **returns** Solution

Input: α, β - two positive integers, the minimum and maximum limits of Є.
Output: Solution

begin
1 **for** Є = α to β **do**
2 **if** (GA_Search (N, K, k, Є, O, I) = Success) **then**
3 **return** Solution
4 **end if**
5 **end for**
end

Algorithm GA_Search (N, K, k, Є, O, I) **returns** Success or Failure

Input: N, K – number of objects and subsets.
k – Population size.
Є – Threshold for fitness.
O, I – positive integers to control the number of iteration.
Output: Success or Failure

begin
1 **while** O > 0 **do**
2 Take Initial Population.
3 **for** i = 1 to I **do**
4 Calculate Fitness value *Fi*
5 **if** F_i < Є **then**
6 Solution := $p_i(j)$, where $f_i(j) = F_i$, *for some j =1 to k*
7 **return** Success
8 **end if**
9 Select k-individuals
10 Do Crossover
11 Do Mutation
12 Select k-best individuals for next generation P_i+1
13 **end for**
14 O := O - 1
15 **end while**
16 **return** Failure
end

4.1. Analysis of the algorithm

Claim 1: The algorithm returns a solution.

Proof: For suitably chosen values of α, β and a finite increment $\epsilon > 0$ of ϵ, the algorithm returns a solution if and only if GA_Search (N, K, k, Є, O, I) returns Success for some value of $\epsilon = \epsilon_i$ within the range and Failure for other $\alpha <= \epsilon < \epsilon_i$. It's easy to see that for a finite value of *O* and *I*, GA_Search will eventually return Failure in case it gets stuck in a local minima after $O(OI)$ iterations. We now prove our claim by contradiction. Let's assume that the GA_Search never returns Success. Referring to equations (1)-(5), the maximum value for $F_i = (K-1)\ C_{av} + C_n - C_{av}$. Setting β = this maximum value and $\alpha = 0$ we can cause line #5 of GA_Search to return Success, which is a contradiction.

Claim 2: The algorithm returns a near-optimal solution.

Proof: Let, the optimum value for $\epsilon = \phi$ and the number of optimum goal states is n_ϕ. Clearly, when $\epsilon < \phi$, GA_Search never returns Success, since $F_i >= \phi$. When $\epsilon > \phi$, we have a more relaxed problem with the number of goal states increased. The more the number of goal states, the more the probability of reaching a goal since the total number of different states remains fixed to *KN*. Suppose GA_Search returns Failure for some $\epsilon_i > \phi$ with probability of failure q_i. Now the probability

that it will fail in the next call with $\epsilon_{i+1} > \epsilon_i$ is q_{i+1} which must be lower than q_i. A drastically increase of goal states after each failure means that q_i will diminish very soon and eventually return a near optimal solution.

5. Experimental Results

5.1. Datasets

Datasets are generated randomly for various combinations of N, K. Datasets are generated such that N is a multiple of K and the optimum D-value is zero. By this way we can easily compute the accuracy of our algorithm since the optimum value is known in advance.

5.2. Results

Table 1 shows a summarized output of the input datasets. Here the average error per subset is calculated to see the accuracy of the algorithm.

5.3. Experimental Facts

We observe some experimental facts-

I) Accuracy increases as N and/or K decreases. (E.g. Test# 1,2,3 & 8,9,10)

II) Accuracy increases as Є decreases. (E.g. Test# 14, 15)

III) Computation time decreases as K decreases (E.g. Test# 8, 9, 10)

IV) Computation time decreases as Є increases (E.g. Test# 14, 15)

Table 1: Summarized output.

Test#	N	K	(ΔЄ, I, O)	Time (sec)	Generations (avg)	Accuracy %
1	10	2	(1, 32, 16)	0	7	100
2	10	4	(1, 32, 16)	0	288	100
3	10	5	(1, 32, 16)	0	1718	99.42
4	15	3	(1, 32, 16)	0	11091	99.44
5	15	5	(1, 32, 16)	1	42138	98.62
6	25	5	(1, 32, 16)	1	20157	97.06
7	30	3	(1, 32, 16)	0	4168	99.56
8	100	5	(1, 32, 16)	4	47513	98.59
9	100	10	(1, 32, 16)	20	196998	93.91
10	100	20	(3, 32, 32)	41	322416	84.27
11	225	9	(2, 25, 50)	35	205956	97.04
12	550	11	(4, 16, 40)	169	239219	95.40
13	1000	25	(150, 32, 32)	20	32195	92.37
14	10000	50	(5000, 16, 32)	26	2048	92.88
15	10000	50	(2500, 16, 32)	42	3328	94.67
16	100000	100	(40000, 16, 32)	33	384	95.89

6. Conclusion and Future Works

In this paper we offered a different approach of applying GA that finds a near optimal solution to the set partitioning problem by iteratively relaxing constraints. Our algorithm needs some parameters to be supplied for a particular instance of the problem. In future we would work on this so that the algorithm itself can set these parameters from experience.

7. References

[1] Fogel, "*Evolving artificial intelligence*", Ph.D. dissertation, Univ. of California, San Diego, CA, 1992.

[2] Fogel, "*Evolutionary Computation: Toward a New Philosophy of Machine Intelligence"*, IEEE press, Piscataway, NJ.

[3] J. Holland, *Adaptation in Natural and Artificial Systems*, University of Michigan Press,Ann Arbor, MI (1975).

[4] P.C. Chu & J.E. Beasley, *"A Genetic Algorithm for the Set Partitioning Problem",*Technical report, Imperial College, 1995.

[5] W.A. Greene, "*Partitioning Sets with Genetic Algorithms*", In J. Etheredge and B. Manaris (eds.), Proceedings of the thirteenth International Florida Artificial Intelligence Research Society (FLAIRS) Conference, May 22-24, 2000, Orlando, FL. (pp.102-106) AAAI Press, Menlo Park, CA.

[6] D. Levine, "*A parallel genetic algorithm for the set partitioning problem*", Ph.D. thesis, Illinois Institute

of Technology, Dept. of Computer Science, May 1994.

[7] D. Powell &M. Skolnick. *"Using genetic algorithms in engineering design optimization with non linear constraints"*. In S. Forrest, editor, Proceeding of the fifth International Conference on Genetic algorithms, pages 424-431 . Morgan Kaufmann, 1993.

[8] Jones, D. R., & Beltramo, M. A. (1991); *"Solving Partitioning Problems with Genetic Algorithms"*, in Belew, K.R. & Booker, L. B. (Eds.), Proceedings of the Fourth International Conference on Genetic Algorithms; Morgan Kaufmann Publ., San Francisco.

[9] Falkenauer, Emanuel (1995); "Solving Equal Piles with the Grouping Genetic Algorithm", in Eshelman, L. J. (Ed.), Proceedings of the Sixth International Conference on Genetic Algorithms; Morgan Kaufmann Publ., San Francisco.