Partitioning Sets with Genetic Algorithms

William A. Greene
Computer Science Department
University of New Orleans
New Orleans, LA 70148
bill@cs.uno.edu

Abstract

We first revisit a problem in the literature of genetic algorithms: arranging numbers into groups whose summed weights are as nearly equal as possible. We provide a new genetic algorithm which very aggressively breeds new individuals which should be improved groupings of the numbers. Our results improve upon those in the literature. Then we extend and generalize our algorithm to a related class of problems, namely, partitioning a set in the presence of a fitness function that assesses the goodness of subsets participating in the partition. Experimental results of this second algorithm show that it, too, works very well.

Introduction

In this paper we first revisit a problem in the literature of genetic algorithms (GAs); the problem concerns arranging numbers into groups so the groups have summed weights that are as nearly equal as possible. For solving this problem we provide a new genetic algorithm which very aggressively breeds new individuals which should be improved groupings of the numbers. We get results which improve upon those in the literature. Then we extend and generalize our algorithm to a related class of set partitioning problems. For this algorithm we have test results to show that it too works very well.

Genetic algorithms are a problem solving paradigm which apply such Darwinian forces as survival of the fittest, mating with crossover of genetic material, and mutation, to the task of finding a solution instance. Implicit here is that candidate solution instances can somehow be represented as sequences of values (usually binary) which mimic the sequencing of nucleotides or genes along a chromosome. Then mating with crossover can take the form of selecting one or several cutpoints along the sequence, identically located for two parents, and exchanging corresponding subsequences of genetic material, to produce a child or two. John Holland (Holland 1975) is credited with inventing the area of genetic algorithms. According to his Schema Theorem, which relies on the linear arrangement of genes, the expectation is that a genetic algorithm will lead towards good solution candidates. A good question is whether linearity of the genes is a *sine qua non* for success of genetic approaches.

The Equal Piles Problem

The Equal Piles Problem for genetic algorithms was defined and first studied by (Jones and Beltramo 1991). It is a problem of partitioning a set into subsets. Given a set of N numbers, partition them into K subsets so that the sum of the numbers in a subset is as nearly equal as possible to the similar sums of the other subsets. (Jones and Beltramo cast the problem in terms of N objects of given heights, which are to be stacked into K piles in such a way that the heights of the resulting piles are as nearly equal as possible.)

This problem is of more than purely academic interest. Its solution is applicable, for instance, to load balancing: given N tasks to be assigned to K processors, how can the tasks be assigned so that the work is evenly distributed?

The particular instance of the problem which Jones and Beltramo investigated had 34 numbers to be partitioned into 10 subsets. The 34 values are reproduced below.

1.	3380	13.	1952	24.	3305
2.	1824	14.	3832	25.	3049
3.	1481	15.	3176	26.	3980
4.	2060	16.	2316	27.	2787
5.	1225	17.	2479	28.	4635
6.	836	18.	3433	29.	4068
7.	1363	19.	3519	30.	2992
8.	2705	20.	1363	31.	5932
9.	4635	21.	1824	32.	528
10.	6481	22.	3305	33.	3304
11.	2588	23.	2156	34.	4107
12.	3380				

Of course, we now know what the ideal subset sum is, namely, the sum of these 34 numbers, divided by 10; this value is 10,000. It turns out that an optimal solution is available for this problem instance. In fact, as noted later by (Falkenauer 1995), several optimal solutions are available, since, for instance, N[9] = N[28], and N[19] = N[20] + N[23] (here, N[j] means the j-th number listed).

Jones and Beltramo tried nine GAs, the best of which turned out to be an ordering GA with PMX crossover (Goldberg 1989). The latter approach, on one occasion, (that is, on one trial, of many generations) came close to finding an optimal solution, but on average its best partitions had an error of 171, where for this paragraph, by error we mean the sum of the absolute values |(a subset's sum) - (the ideal subset sum)|.

Falkenauer picked up this problem in (Falkenauer 1995). Jones and Beltramo cast their work in terms of chromosomes of length 34, for the 34 individual numbers being grouped. Falkenauer, on the other hand, argues that for a "grouping" problem such as this one, an entire subset should be treated as a gene. That is, where possible, manipulate entire subsets versus the numbers in them. In particular, when parents exchange genetic material, they should begin by exchanging entire subsets. Falkenauer also notes that the order of subsets within the chromosome is immaterial; put another way, arranging genetic material in a linear sequence has no natural persuasion for this problem. Falkenauer runs his Grouping Genetic Algorithm (GGA) on this problem, and does markedly better than Jones and Beltramo. In 30 trials, each of up to 3500 generations, he finds the optimal solution on 26 of the 30 trials, after an average of 17,784 (not necessarily different) partitions have been encountered. Then making his crossover operator greedier, he improves upon himself, finding the optimal solution on all 30 trials, after an average of 9,608 partitions have been encountered.

The Solution

Our solution to this problem is akin to that of Falkenauer, but differs from it in distinct ways, most notably in the crossover operator, and the mutation practiced is completely different as well.

We represent a subset as an array of 34 boolean values, with the obvious interpretation that the j-th component of the array equals true if and only if the j-th value is included in the subset at hand. This is more space-expensive than other approaches, but results in time economies. The *error of a subset* is the absolute value of the difference between the sum of the numbers in the subset versus the ideal subset sum of 10,000. A *partition* is then a list of 10 subsets. It is immaterial in the abstract what order subsets are listed in the partition, but for our purposes we list them in increasing order of error, so that the more accurate subsets appear first. The *error of a partition* is then the Euclidean norm (square root of the sum of squares) of the vector of errors of its 10 subsets. A *population* is a set of partitions. Let us say for now that population size will be 100. For our purposes, the individual partitions in a population are kept arranged in increasing order of error. *Fitness* will be the complement of error: the more erroneous a partition is, the less fit it is. Darwinian forces are brought to bear when the n-th generation of a population is transformed into the (n+1)-st generation, as next described. The descriptions given are for our most successful efforts.

Survival of the fittest surfaces in two forms. Firstly, *elitism* is practiced: a (small) percentage (seven percent) 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. Specifically, the errors of the individual partitions in a population range from some Low value to some High value. The fitness of a partition is deemed to be (High + 1 - (own error)). Then a partition is chosen for parenting with a probability equal to its proportional fitness, that is, (own fitness) / (sum of fitnesses).

Crossover

Mating with crossover, which produces one child in our approach, is done in such a way as to very aggressively accumulate good subsets into a child partition. For this reason we name our algorithm Eager Breeder. It is described as follows. Establish a pointer at the beginning of the list of subsets of parent-1 (start with the best of the subsets). Establish a like pointer for parent-2. Of the two parental subsets now pointed at, copy the less erroneous one into the list of child subsets (flip a coin if the subsets have equal errors), and advance the corresponding parents pointer. However, never copy a subset into the child if that subset already exists in the child (because it was earlier acquired from the other parent). Keep copying parent subsets until the child has 10 subsets. Note that several subsets from parent-1 may be copied into the child before one from parent-2 is copied. Conceivably, the child is identical to one of its parents. The subsets acquired from parent-1 are disjoint (they were in the parent), and the same is true of those acquired from parent-2. Thus, for each of the 34 values, it can be said that value appears in at most two of the subsets in the child. If a value appears in two child subsets, remove it from the more erroneous subset (flip a coin if the subsets

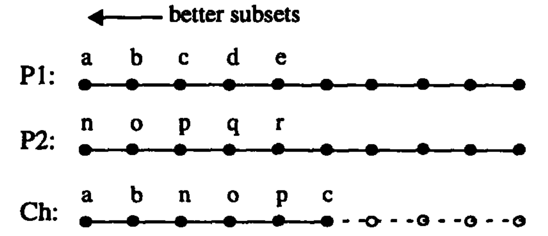

Figure 1: Only the best subsets (genes) from parents P1 and P2 enter the child.

have equal errors). If a value appears in one child subset, then that is desired and no adjustment is needed. Collect together the values that as yet appear in no child subset at all. Distribute them into the child subsets, following the principle to put the biggest as-yet-unassigned value into the subset which currently has the lowest sum.

Generational Change

Generational change is accomplished as follows. As said earlier, elitism makes a small percentage of individuals in the current generation automatically survive into the next generation. The rest of the population in the next generation is obtained by mating with crossover, following a weighted roulette wheel selection of parents for mating. Once the population in the next generation is up to the original population size, we sort it into increasing order of error, then individuals in the next generation are subjected to degrees of mutation. After the mutation phase, the new population is again sorted into increasing order of error.

Mutation

One stochastic mutation step is performed as follows. If a given probability is met, then one of the 34 values is selected at random, removed from the subset it is in, and added to a randomly chosen subset (conceivably the same one it was just in). The newly formed population is subjected to mutation by degrees, with more erroneous individuals undergoing more mutation. The following description is illustrative. A small number of individuals (equal to the number who survived under elitism) undergo one mutation step with a 10% chance. Up to the next four-tenths of the population each undergo 4 mutation steps, with each step occurring with 50% probability (thus we expect each individual in this band to undergo 2 actual mutation steps on average). Up to the next seven-tenths of the population undergo 10 mutations steps, each with 50% probability. The remaining individuals each undergo 20 mutation steps, each with 50% probability. In general, generous mutation on the population's least accurate individuals was found to work well.

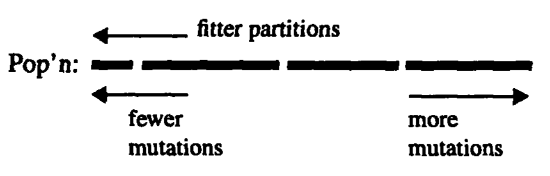

Figure 2: Graduated mutation rates

Results

Now we describe our results. Experiments with different parameter settings have different outcomes. Perhaps our best was the following experiment. Population size is 250, there are 30 trials, each of which is allowed to run to 40 generations. An optimal partition was found on 29 of the 30 trials; on the other trial, the best individual found was the best sub-optimal solution. A best individual was found on average generation number 12.97, implying that approximately 3,242 partitions were encountered. These results are almost as "perfect" as those of Falkenauer. As his approach encounters 9,608 individuals on average, our approach has 33.7% of the cost of his.

	Falkenauer's Greedy GGA	Eager Breeder
Population size	50	250
Max # generations	3500	40
Number of trials	30	30
Trials finding optimal partition	30	29
Average # individuals to find optimal partition	9,608	3,242

Table 1: Comparison of past and present research.

By comparison, the crossover done by Falkenauer randomly chooses some piles from parent-1, and adds those to the piles of parent-2. Any piles originating from parent-2 which overlap with a pile originating from parent-1 are eliminated, their values are collected, some of these values are used to start enough new piles, then remaining values are distributed, following the principle to put the biggest as-yet-unassigned value into the subset which currently has the lowest sum. His greedy crossover is similar, except that the piles first chosen from parent-1 are pared of their less accurate members. His mutation operation consists of emptying a randomly chosen pile, joining to those values enough more out of remaining piles to reach one-fourth of the values at hand, then using those to start a new pile, and finally distributing remaining as-yet-unassigned values by the now familiar heuristic.

The Extension

We wanted to extend our ideas to related but more general set-partitioning problems. We will retain the characteristic that the domain provides us not just a measure of the fitness of a partition but rather the fitnesses of its individual subsets. For our approach this fitness measure of a subset could be a real number in the unit interval [0, 1], for example.

Intuition suggests that it is more challenging to correctly partition when the partitioning subsets are of quite diverse sizes. So, we set ourselves a manufactured problem, in which 1 subset has 20 (specific) elements, 1 subset has 10 elements, 2 subsets have 5 elements each, 3 subsets have 2 elements each, and 5 subsets consist of just 1 element each. That makes 12 subsets altogether, from 51 elements. For representation, once again a subset is an array of (51) boolean values, and a partition is a list of (12) subsets.

Next we describe the fitness function for our domain. Our fitness function will be defined in a way that is sharply tuned to the problem at hand. On the other hand, each value it returns merely reports a measure of the satisfactoriness of a candidate subset. Suppose that subset S contains element X. We ask, how well does S co-associate the elements which are supposed to be included in or excluded from the subset which contains X. With regard to element X, subset S can be rated from 0 to 50, where 0 means worst possible (for each of the 50 other elements, S fails to include those that should be in the same subset as X, and S incorrectly includes those that should be excluded from the subset that includes X) and 50 means best possible. Call this the *co-association rating of S with regard to X*. Then we define the *co-association of S* to be the average of the co-associations over the elements X which S contains. Define the *error of S* to be the difference between its desired co-association of 50 and its actual co-association; error is in the range [0.0, 50.0]. Define the *fitness of S* to be 50 minus its error. Note that the fitness of a subset does not reveal at all what elements belong in the subset; instead, it measures how closely the subset comes to being one of the targeted ones.

As before, we arrange the 12 subsets in a partition in increasing order of error. We define the *error of a partition* to be the Euclidean norm (square root of sum of squares) of the vector of errors of the subsets in the partition. A population will be a set of partitions; population size remains constant over generations.

The Darwinian forces now brought to bear on the population are a carryover from our earlier work, with one important difference. Above, under mating with crossover, recall that when child subsets are culled from the best of those of the parents, those subsets might not yet include all the numbers in the set being partitioned into piles. And above, when our goal was to build piles of equal sums, we distributed as-yet-unassigned set elements by following the principle to loop, putting the biggest one into the subset which currently has the lowest sum. For our new problem, an analogous approach would be to loop, putting a randomly chosen as-yet-unassigned set element into the subset which currently is the most erroneous. On a typical run of 30 trials, this approach discovered the target partition on 7 of the 30 trials, and for the other 23 trials, the best partition tended to make the error of lumping together the two largest subsets, of sizes 20 and 10 (this introduces an error of 13.33).

Our modification is as follows. As before, accumulate child subsets by culling the best of the subsets out of the parents; if an element is in two child subsets, remove it from the more erroneous one. Now form a weighted roulette wheel, based upon the errors of the child subsets at hand so far. Round up the as-yet-unassigned set elements, and distribute them into the child subsets, by using the weighted roulette wheel to favor the more erroneous subsets for reception of an element. In short, this introduces stochastic variety into which subset receives an as-yet-unassigned element.

Our results are as follows. Population size was set at 100. On 30 trials of 200 generations, the targeted partition was discovered on all 30 trials, on average generation number 48.7. (Other trials which varied the problem parameters sometimes missed the targeted partition on some of the trials.) These are very good results, when one considers the fact that there are a stupendous number of different ways to partition 51 elements into 12 non-empty subsets. By our computerized count there are around 1.97E+46 such ways. Our algorithm discovered the target partition after encountering 4,870 partitions.

As a last experiment, we used this algorithm on a partitioning problem where the subsets were not of diverse sizes. For this last experiment, there are 8 subsets, each of 6 (specific) elements. With population size set at 100, on 30 trials of 200 generations, the targeted partition was found on all 30 trials, on average generation number 20.9. It is not so surprising that finding a partition with very diverse subset sizes is more costly.

Subset Sizes:	diverse (20,10,5,5,2, 2,2,1,1,1,1,1)	same (6,6,6,6, 6,6,6,6)
Number of trials	30	30
Trials finding optimal partition	30	30
Average # individuals to find optimal partition	4,870	2,090

Table 2: Performance of Extended Eager Breeder. In all trials, population size = 100, and the maximum number of generations = 200.

Conclusion

Our algorithm Eager Breeder, for solving the Equal Piles Problem, is an incremental improvement upon Falkenauer, with almost identical accuracy but one-third the cost. The extension of our algorithm, to discovering a targeted partition of 51 elements into 12 subsets, also had impressive results.

Genetic algorithms are a general problem-solving approach. The incorporation of problem-specific heuristicism can improve the performance of this approach. In our research, the genetic paradigm has been tailored to suit the case that we are building set partitions. When a child is formed from two parents by crossover, only the best genes (subsets) from the two parents enter the child, and even then a gene is excluded if a copy is already in the child.

References

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.

Goldberg, David (1989); Genetic Algorithms in Search, Optimization, and Machine Learning; Addison-Wesley Publ., Reading, MA.

Holland, John (1975); Adaptation in Natural and Artificial Systems; University of Michigan Press, Ann Arbor, MI.

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.