Pakistan Journal of Applied Sciences 2(1): 71-73, 2002

A Genetic Algorithm to Solve the Maximum Partition Problem

Wael Mustafa
Computer Science Department
An-Najah National University, Nablus, Palestine

Abstract: A *maximum partition* of a directed weighted graph is partitioning the nodes into two sets such that it maximizes the total weights of edges between the two sets. In this study a genetic algorithm is proposed to solve the maximum partition problem. Experiments performed on randomly generated graphs of different sizes show that the proposed algorithm converges to an optimal solution faster than the existing heuristic algorithm.

Keywords: Genetic algorithms, Text compression, Optimization, Graph Algorithms

Introduction

The work of Manber (Manber ,1997) , which focused on a compression method that allows fast search directly on compressed text, introduced a new graph problem. The problem is determining which pairs of characters to be replaced by single unused characters. Pairs of characters should be chosen such that more compression is achieved. Additionally, to make direct search through the compressed text possible, pairs cannot overlap (i.e. the first character in one pair cannot be the second character in another pair).

The problem above is abstracted as a graph problem, known as the Best-Non-Overlapping-Pairs Problem in (Manber ,1997) . A directed graph G = (N, E), is constructed where the nodes correspond to unique characters used in the text. Edges represent character pairs and weights represent the frequencies of these pairs in the text being compressed. We want to partition the nodes of the graph into two sets N1 and N2, such that the sum of weights of edges from N1 to N2 is maximized. Fig 1 shows an example of a directed graph and its maximum partition with sum of weights from N1 to N2 being 19. The best non overlapping pairs according to this partition are bc, bd, and ac.

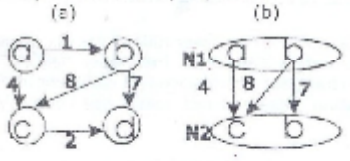


Fig. 1: A directed weighted graph (a) and its maximum partition (b)

The work in (Manber ,1997) also presented a heuristic algorithm (HA) that produces near optimal solutions for the maximum partition problem. This HA initially partitions the nodes randomly into two sets N1 and N2. Next, each node is examined to see whether switching this node to the other set would improve the total sum of weights from N1 to N2. Such switches are continued until no more are possible. The process is repeated a number of times with different random seeds and the best result is reported.

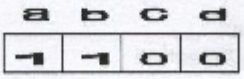
The maximum partition problem for a graph with *n* nodes has a solution-space size of 2^n, corresponding to 2^n subsets. It is also an NP problem (Manber ,1997) . NP problems can only be solved in polynomial time by non-deterministic algorithms (Horowitz and Sahni,1978) . Numerous heuristic methods have been developed to solve such problems, but none of them promise optimal solutions.

This research presents a genetic algorithm (GA) to solve the maximum partition problem. Genetic algorithms are able to cover large search spaces effectively and have been very successful in producing near optimal solutions to NP problems. A genetic algorithm is characterized by creating solutions through combining parts of different solutions and making small mutational changes to solutions [Davis,1991; Goldberg, 1989; Holland,1975;Mitchel ,1998]. In this paper we take the advantages of GAs to solve the maximum partition problem .

Genetic Algorithms: Genetic algorithms (GAs) are search algorithms modeled after the behaviors of genetic processes in nature. Genetic algorithms operate on a population of individuals, called chromosomes. A *chromosome* is a string of characters, called *genes* and represents a possible solution in the search space of the problem. The quality of this solution is called chromosome fitness. The fitness of each chromosome is computed according to a problem-dependent fitness function.

A *generation* is a GA step in which several events occur. A number of chromosomes with worst fitness value are removed from the population. These are replaced by new chromosomes obtained from applying crossover operations to the remaining chromosomes in the population. *Crossover* is the operation of exchanging corresponding genes between two chromosomes. In order to achieve diversity in the population and to prevent the algorithm from converging prematurely (i.e. before reaching the optimal solution), mutation is applied to every chromosome in the population. *Mutation* is the operation of changing chromosome genes randomly with certain probability. Excluding most fit chromosomes in the population from mutation is known as elitism.

The Genetic Algorithm to Solve the Maximum Partition Problem: Following is a definition of a GA to solve the maximum partition problem for a directed graph with set of nodes N. A partition of this graph into two sets of nodes N1 and N2 is represented by a chromosome of |N| bits. The *i*th bit indicates whether node *i* is in N1 or N2. As an example, the following



chromosome represents the maximum partition shown in Fig. 1. The fitness of a chromosome is the sum of weights from N1 to N2 in the partition it represents. For example, the fitness of the chromosome above is 19.

The GA to solve the maximum partition problem is as follows.

Algorithm: Maximum Partition (G(n,e): Weighted Directed Graph):

1. Initialize the population with random chromosomes.
2. Improve fitness of the initial population by applying the HA of (Manber ,1997) to (*population-size*/2) chromosomes selected randomly. For each such chromosome, every bit is examined to see whether flipping this bit would improve the fitness of the chromosome. This process is repeated until no more flipping is possible in the chromosome.
3. Repeat K times (generations).

a. Compute the fitness value for each chromosome in the population.

b. Sort the chromosomes in descending order according to fitness values. The population is then divided into a lower half and an upper half, based on fitness values. The upper half contains chromosomes with the higher (better) fitness values.

c. Remove 20% of the chromosomes from the population. These chromosomes are selected randomly from the lower half of the population.

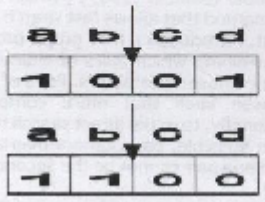
d. Use crossover to construct new chromosomes to replace the removed ones. Two parent chromosomes are randomly chosen from the upper half of the population. The first |N|/2 bits of one chromosome are concatenated with the last |N|/2 bits of the other chromosome and vice versa. This creates two new chromosomes. Crossover is performed (*population-size*/10) times in order to produce (*population-size*/5) new chromosomes.

e. Apply mutation to all chromosomes in the population except the chromosome with the highest fitness value and the chromosomes created by crossover in step d. Each bit is considered when a chromosome is mutated. If a chromosome is in the lower half of the population, the bit is flipped with a probability of 0.3. If a chromosome is in the upper half of the population, the bit is flipped with a probability of 0.01.

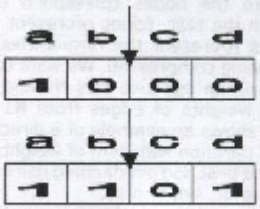
f. Remove duplicate chromosomes from the population and replace them with randomly generated chromosomes.

4. Output the highest fitness chromosome in the population.

Experiments show that the GA above is more likely to converge to an optimal solution in small number of generations when the initial population contains some chromosomes with good fitness values. The first two steps of the algorithm make the initial population by creating random chromosomes and then improving half of these chromosomes using the HA of (Manber ,1997) Experiments show that improving more than half of the initial population does not improve the GA chances of converging in a small number of generations. The third step in the algorithm determines the next generation. After sorting chromosomes in the current population according to fitness values, 20% of this population are randomly chosen from the lower half to be removed. These are then replaced by new chromosomes obtained by applying single point crossover (Davis, 1991) to chromosomes selected randomly from the upper half of population. The crossover operation is illustrated in the following diagram.



Produces the new chromosomes:



Mutation is applied to the lower half of the population at much higher rate (30%) than the upper half (1%) to make more changes to chromosomes with lower fitness. Other mutation rates did not make the GA converge faster.

Results

We experimented with various population sizes on many graphs. The algorithm converged to an optimal solution, in a short execution time more often when setting the population size to approximately $(2/3)n$ where n is the number of nodes in the graph. Smaller populations resulted in longer time for the algorithm to converge. Larger populations did not improve the results any further.

To evaluate the performance of the presented algorithm against the existing HA of (Manber ,1997) we have implemented both algorithms using C on an 866 Pentium III processor. To ensure algorithm efficiency in both space and time, graph partitions in the HA and chromosomes in the presented GA were implemented as unsigned integers. Each 16-bit unsigned integer

represents a set of size 16 where every bit contains information about one element. Sets of more than 16 elements are represented as arrays of unsigned integers. This representation allows performing sets operations in the two algorithms efficiently using bit-wise operators. The two algorithms were applied to 3 complete graphs (i.e. there is an edge between every pair of nodes) with number of nodes 32, 64, and 128. The weights of edges were chosen randomly from the range [0, 32767]. The two algorithms were run on each graph for different time periods. In each run, the algorithms were applied a number of iterations until the time period expired. Each run in a given period was repeated a number of times using different random seeds. The number of times the algorithms converged to an optimal solution is reported.

Table 1: Results for running GA and HA 50 times on a 32-nodes graph for different time periods

Run- time	# Optimal	
(Sec)	GA	HA
5	38	28
15	45	42
25	49	49
35	50	50

Table 1 shows results for the 32-nodes graph. Each of the two algorithms was run on this graph for the time periods: 5, 15, 25, and 35 seconds. Runs of the two algorithms were repeated for each of these periods 50 times. Our genetic algorithm (GA) converged to an optimal solution 26% more than HA converged to an optimal solution when run-time period was 5 seconds. As execution time increased, the difference between convergence of the algorithms decreased. Both algorithms converged to optimal solutions in all of the 50 runs when execution time increased to 35 seconds.

Table 2:Results for running GA and HA 50 times on a 64-nodes graph for different time periods

Run- time	# Optimal	
(Sec)	GA	HA
100	21	13
200	37	26
300	44	34
400	43	41
500	45	40
600	48	48

The two algorithms were run on the 64-nodes graph also 50 times for each of the time periods 100, 200, 300, 400, 500, and 600 seconds. The results are shown in Table 2. The GA converged to an optimal solution more than HA by about 61% when the algorithms were run for 100 seconds. The GA also converged to an optimal solution more than HA by about 42% for the 200-seconds execution time period. When run-time was increased to 300 seconds, the GA converged more than the HA by about 29%. The difference in convergence times became smaller when the two algorithms executed for a larger time period. Both algorithms converged in 48 out of the 50 runs when execution time was set to 600 seconds. Table 3 shows the results for the 128-nodes graph. Due to large execution times for this graph, which contains 128 X 128 edges, runs were repeated only 20 times. The execution time periods for this graph were 2000, 3000, 4000, and 5000 seconds. The GA converged to an optimal solution more than HA by about 70% when the two algorithms ran for 2000 seconds. As with the other two graphs, the difference between convergence times decreased as execution time increased. Both algorithms converged in most of the 20 runs when execution time reached 5000 seconds Finally.

Table 3: Results for running GA and HA 20 times on a 128-nodes graph for different time periods

Run- time	# Optimal	
(Sec)	GA	HA
2000	12	7
3000	14	10
4000	15	14
5000	18	16

Conclusions

We presented an algorithm to solve the maximum partition problem. This problem arises in text compression where the aim is to find character pairs to be replaced by unused characters in such away that the compressed text can be searched directly. The presented algorithm combines a heuristic method introduced in (Manber, 1997) and genetic algorithms. Results of running the algorithm on random graphs how that it converges to an optimal solution faster than the existing heuristic algorithm. A sample of these results was given above. We believe that the ability of the algorithm to converge to an optimal solution faster is very valuable especially in text compression applications. This is even becomes more important when there is a need to apply the algorithm frequently in order to determine different character codes for different text parts.

Future work is needed to investigate the effect of using the heuristic in (Manber ,1997) with GAs after the initialization step. One possibility, for example, is to apply this heuristic on part of the population after every certain number of iterations. Other future work is needed to incorporate the presented algorithm with the text compression method in (Manber ,1997) . In particular, the algorithm needs to be tested on graphs that represent characters distribution in real texts. The impact of the resulting character codes on both space and time need to be investigated.

References

Davis. 1991. Handbook of Genetic Algorithms. Van Nostrand Reinhold, New York.

Goldberg. 1989. Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesly.

Holland. 1975.Adaptation in Natural and Artificial Systems. University of Michigan Press.

Horowitz and Sahni. 1978. Fundamentals of Computer Algorithms. Computer Science Press, Inc., Rockville, MD.

Manber. 1997. ACM Transactions on Information Systems, Vol. 15, No. 2, Pages 124-136.

Mitchell. 1998. An Introduction to Genetic Algorithms. MIT Press.