Chapter 1

Introduction

The last thing one discovers in writing a book is what to put first.
Blaise Pascal

Mathematics provides formal and exact solutions to the real world problems as well as problems of abstract nature. In the early stages it provided laws for manipulating discrete quantities of objects, later, the laws were extended for continuous amounts. Many disciplines of mathematics started with abstraction but later found extensive use in real world, directly or indirectly. An example of such a branch is complex algebra. Graph theory on the other hand started with an application, later on many branches evolved in the area as a result of abstraction. However, these extensions in turn have been applied in more complex real world problems or in the problems emerged due to evolution of computers. Graph theory shifted the focus again to discrete mathematics as entities involved in it are discrete.

The origin of graph theory dates back to 1736 when Léonhard Euler, a Swiss mathematician presented the solution to Königsberg bridge problem. But in the past 40 or 50 years, it became an important discipline in computational mathematics and other related fields. After the resurgence of graph theory in the era of computers, the evolution of computers and

the graph theory complemented each other. For example, chip design for very large microprocessors wouldn't be possible without the graph theory, yet that same microprocessor will power a computer to process a large graph.

1.1 Graph and Grouping Problems

A graph $G = (V, E)$ consists of a finite non-empty set V, called the vertex set, and a set E, called the edge set, of ordered or unordered pairs of distinct elements of V. An element of V is called a vertex or a node or a point whereas an element of E is called an edge. An edge that is associated with an ordered pair of vertices is called a directed edge whereas an edge that is associated with an unordered pair of vertices is called an undirected edge. A graph in which every edge is directed is called a directed graph. A graph in which every edge is undirected is called an undirected graph. If some edges are directed and some are undirected in a graph then the graph is called a mixed graph.

It is possible to model a number of real world problems by means of graphs. For example, the World Wide Web can be modelled as a directed graph where each node is a web page and each hyperlink is an edge. Study of web graphs provides insight into a lot of things, such as algorithms for crawling, searching or ranking web resources. In stochastic process theory, a Markov chain is a graph in which events are vertices and a positive probability of direct succession of two events is a directed edge connecting the corresponding vertices. Many problems in chip design can also be modelled as graph problems.

The application area of graph theory is so vast that it spans almost every discipline, viz. military problems, manufacturing, civil engineering,

computer and telecommunication networks, electrical networks, molecular biology, economics etc. in addition to the fields described already.

In this thesis we are concerned with undirected graphs only. Hence we will define the terminology used in this thesis in the context of undirected graphs. If e is an edge then $e = \{u, v\}$ (we have used $\{u, v\}$ and (u, v) interchangeably as only undirected graphs are considered in this thesis) where u and v are two distinct elements of V (called the end points or the end vertices of edge e). The edge $e = \{u, v\}$ joins (or connects) vertices u and v. Instead of writing an edge e as $\{u, v\}$ or (u, v), we can also write it as uv. In case of undirected graph edge uv is same as edge vu. An edge is incident to each of its end points. Two vertices are adjacent if they are joined by an edge. Two edges are adjacent if they have a vertex in common.

A graph is called a weighted (or edge-weighted) graph if non-negative weights are associated with every edge of the graph. If weights are associated with every vertex of the graph then the graph is called a vertex-weighted graph. A graph is called a complete graph if every vertex is joined by an edge to every other vertex of the graph. A graph is called a planar graph if it can be drawn in a plane in such a way that no two edges intersect geometrically except at a vertex to which both are incident. A graph $G' = (V', E')$ is called a subgraph of the graph $G = (V, E)$, if and only if $V' \subseteq V$ and $E' \subseteq E$.

Given a graph $G = (V, E)$, a walk in G from v_o to v_m is a finite sequence of edges of the form $v_0v_1, v_1v_2, \ldots, v_{m-1}v_m$ in which any two consecutive edges are either adjacent or identical. Such a walk determines a sequence of vertices $v_0, v_1, \ldots, v_m$. The vertex v_o is called the initial (or starting or originating) vertex of the walk and vertex v_m is called the final (or ending or terminal) vertex of the walk. The number of edges in a walk is called its length. A walk in which all edges are distinct is called a trail. A trail is

called a path if all vertices $v_0, v_1,..., v_m$ are distinct (except possibly $v_o = v_m$). A path or trail is closed if $v_0 = v_m$. A closed path is called a cycle.

A graph is called connected if given any two vertices of the graph there is a path from one vertex to the other. A tree is a connected graph that has no cycles. A subgraph $G' = (V', E')$ of the graph $G = (V, E)$ is called the spanning-tree of G if and only if G' is a tree and $V' = V$.

Here it is to be noted that terminology used for graph theory is not standard. Our definition of graphs precludes parallel edges (more than one edge between a pair of vertices) and loops (edges incident to only one vertex) whereas many authors allow them in the graph. Some authors have used the term simple graph for our definition of graphs. We have chosen the definition of graphs which is most suitable for describing the problems attempted in this thesis.

Like graph problems, there is another class of interesting problems called grouping problems, which also has its roots in discrete mathematics. Grouping problems involve partitioning a set U of objects into a collection of mutually disjoint subset u_i of U. Grouping problems are also of practical importance. For example, the problem of task allocation in a multiprocessor system is a grouping problem. Map coloring problem is another example of a real world grouping problem.

In many graph and grouping problems, some hard constraints are imposed to rule out invalid constructions. These constraints filter out trivial solutions in most of the cases and make these problems hard.

1.2 Graph and Grouping Optimization Problems

Many graph and grouping problems are optimization problems and most of these are NP-Hard. NP-Hardness of a problem means that any known

exact algorithm for the problem will run in time that grows exponentially with the size of problem instance and no polynomial time verification algorithm is known to check the optimality of a proposed solution. Therefore, it is not possible to solve every arbitrary instance of arbitrary size of a NP-Hard problem with the help of an exact algorithm. However, for a specific type of instance of arbitrary size or for every instance of small size, we may find an exact algorithm fast enough to produce the solution in a reasonable amount of time. However in many real world problems the size of the instance is so large that no exact algorithm can be applied. These are the situations where we look towards approximation techniques. In this thesis, we attempt some of the NP-Hard graph and grouping optimization problems through particular type of approximation techniques.

1.3 Approximation Techniques

If there is no upper limit on the size of instances that have to be handled by an exact algorithm for a NP-Hard problem or if the known best exact algorithm is unable to handle the intended instances in a reasonable time, the search for the optimum has to be abandoned. At first thought this appears to be sad, however, most practical problems do not require optimality. Instead, only a good solution to the problem is desired. Approximation techniques are recommended in this case. These techniques, though do not guarantee to reach the global optimum, are nevertheless good enough to reach near optimal solution in most of the cases. These techniques usually take polynomial time. Here it is to be noted that the solution returned by approximation techniques is near to the optimum solution in terms of cost function but may be widely separated from global optimum in search space. Usually the search space in NP-Hard problems is rugged and consists of a large number of local optima, many of which are of comparable quality to global optimum.

There are NP-Hard problems for which very good approximation techniques are known while some other NP-Hard problems do not have any reasonably good approximation technique. Here a word of caution is worthwhile. Many NP-Hard problems are closely related and an instance of one problem can be transformed in polynomial time into an equivalent instance of another problem such that an optimal solution of transformed instance automatically determines the optimal solution of original instance. Therefore one may be tempted to obtain a good approximate solution of an instance of a problem for which no good approximation technique is known by transforming the instance into an equivalent instance of another problem for which a good approximation technique is known. However this strategy may be counter-productive in many cases. Garey and Johnson (1979) showed that a good approximation of optimum in one problem can lead to a disastrously bad solution in another problem even though the instances being solved are tightly related.

There are mainly two categories of approximation techniques– Heuristics and meta-heuristics. Heuristics are basically "Rules of Thumb" recipes for solving a particular problem based on common sense. A heuristic approach usually involves employing some piece of knowledge about the structure of the problem under consideration, to devise a strategy for solving it. In general, a heuristic finds pretty good solutions but there is no proof that the solutions could not get arbitrarily bad. Meta-heuristics such as genetic algorithm, simulated annealing, tabu search are those heuristics whose applicability is not limited to a particular problem but to a wide range of problems. However, many meta-heuristics require that problem is represented in a form which is suitable for their application. A third category of approximation techniques is approximation algorithms, which are nothing but heuristics with proven solution quality. However, many graph and grouping optimization problems are so hard that it is impossible to

obtain good polynomial time approximation algorithms for them. Therefore we are left with no choice except using heuristics and meta-heuristics. In many such cases a hybrid approach is used to obtain a good approximate solution to a NP-Hard problem, where a meta-heuristic is combined with a problem specific heuristic. This thesis is particularly concerned with one meta-heuristic called genetic algorithm which falls under the broad class of evolutionary algorithms. All but one problem attempted in this thesis are solved using the hybrid approach combining the genetic algorithm with a problem specific heuristic.

1.4 Evolutionary Algorithms

Conventional optimization techniques more or less rely on auxiliary information about the objective function such as its gradient, hessian, linearity and continuity etc. to choose the next point in the search space. This dependence limits their domain of application to the functions which are uni-modal, continuous and differentiable. However, many functions are multi-modal, discontinuous and non-differentiable. Stochastic sampling methods have been used to optimize these functions where the next sampled points are chosen based on stochastic sampling/decision rules rather than a set of deterministic rules.

In the recent past there has been a focus on the stochastic algorithms for search and optimization that in essence rely on the biological concepts of evolution. These algorithms are given the umbrella term evolutionary algorithms. All evolutionary algorithms are based on the evolution of a population of potential solutions to a certain problem. The population of possible solutions evolves from one generation to the next based on a "survival of the fittest" strategy, ultimately arriving at a satisfactory solution to the given problem. These algorithms differ in the way the new popula-

tions are generated and in the way the members are represented within the algorithm. The leading branches of evolutionary algorithm are genetic algorithm, genetic programming, evolutionary programming and evolutionary strategies.

1.5 Genetic Algorithm

The most prominent among the evolutionary algorithms is genetic algorithm. Genetic algorithm (GA) was proposed by John Holland in 1960s and later developed by Holland and his co-worker in 1960s and 1970s. His original efforts were to study and simulate evolutionary adaptation as it occurs in nature. Later genetic algorithm was used to solve various kinds of problems including search and optimization. The main reason for the success of genetic algorithm is its robustness and its ability to explore several possible areas of search space at the same time. Genetic algorithm provides implicit as well as explicit parallelism. The implicit parallelism is inherent in the algorithm. This fact was propounded and proved by John Holland (1975) in his famous schema theorem. Explicit parallelism may arise if different individuals in the population are manipulated and evaluated in parallel. The ability of the genetic algorithm to mimic evolution and therefore enabling solutions to adapt according to environment makes it very useful for search, optimization and machine learning.

As genetic algorithm uses a number of terms from biology, we think it is appropriate to define these terms before any further discussion. Table 1.1 defines the most commonly used terms in genetic algorithm.

An individual solution in genetic algorithm is represented by a set of parameters. These parameters are considered as genes that in turn form a chromosome or individual, when structured as a string. The genetic algorithm begins with a population (the size of which may depend upon the

Table 1.1: Most commonly used terms in a genetic algorithm

Phenotype	The potential solution to the problem
Chromosome	The representation of the phenotype in a form that can be used by the genetic algorithm (generally as a linear data structure)
Genotype	The set of parameters encoded in the chromosome
Gene	The unalterable pieces of data constituting a chromosome
Phene	A trait in the phenotype corresponding to a gene
Alleles or Alphabet	The set of values a gene can take on
Population	The collection of chromosomes that evolves from generation to generation
Generation	A single pass from the present population to the next one
Fitness	The measure of the performance of an individual chromosome on the actual problem
Evaluation	The translation of the genotype into the phenotype and the calculation of its fitness
Genome	The complete collection of genes of an individual
Locus	Particular position on the chromosome
Problem space	The space comprising of all the possible solutions of the problem under consideration
Representation Space or genotype space	The space comprising of all the possible genotypes of the problem under consideration
Fitness landscape	A representation of the space of all possible genotypes along with their fitness
Solution space or Search space	Some collection of candidate solutions to the problem with some notion of a "distance" between them. This is a subset of problem space.

problem) of such chromosomes. Normally all the individuals in the population are randomly generated to have an unbiased representation of the search space. However, if we know beforehand or otherwise some limited regions of the search space, which contain good potential solutions, then these can be used to start the genetic algorithm so that the available information about the fitness landscape is exploited to get a better start.

Every solution (chromosome) is assigned a positive score which is called its fitness and is a measure of how much better the solution is in comparison to others. To find out the fitness of an individual, one may need to convert it first to the phenotype. The fitness may be evaluated then through the interaction of this phenotype with its environment which may also involve the neighbourhood of this individual. The module which is used for optionally converting a genotype to phenotype and evaluating the fitness is called evaluation function.

After assigning fitness values to each member of the population, it is checked whether termination criteria is satisfied. The termination criteria may be a time constraint, computational steps constraint or some value of the fitness which is to be achieved. If a termination criterion is satisfied then algorithm is terminated at once otherwise we make a mating pool out of this population by using a selection method on the population. The selection method may pick individuals for mating pool from the current population in such a way that better individuals may get more than one copy whereas some bad ones may not get selected at all. This is done to emphasize fitter individuals in the mating pool in the hope that their offspring will in turn have even higher fitness.

The individuals in the mating pool are allowed to reproduce through sexual or asexual process. The sexual process is called crossover or recombination. Recombination means mixing the genes of two or more individuals (called parents) and from them generating one or more new individuals (called offspring). Usually only two individuals are recombined to produce one or two offspring. Each of these offspring will have some genes from one parent and rest from the other. In the asexual process there is only a single parent and the offspring produced is an exact copy of its parent. The offspring, irrespective of the process it is generated, will undergo mutation. Mutation means random change in gene values or information con-

tent of a chromosome relevant to the problem under consideration. Both mutation and crossover occur with certain probabilities called mutation rate and crossover rate. Mutation rate decides how many genes of a chromosome will be mutated and crossover rate determines what fraction of old generation will go to the next generation through crossover or sexual process. As crossover and mutation occur with some probability therefore some population members can go to next generation unaltered.

As a result of reproduction, subsequent population is generated; this is called next generation of the population. The new generation is now evaluated. Termination criterion is tested again and if it is satisfied the algorithm is terminated at once else the genetic algorithm will continue till the termination condition is satisfied.

The genetic algorithm described in this section will be referred to as simple genetic algorithm to differentiate it from the permutation and grouping genetic algorithm to be discussed in subsequent sections.

1.5.1 Theoretical Foundations of Genetic Algorithm

The theoretical foundation of genetic algorithm has been provided by the schema theory of Holland (1975). The general idea is that a chromosome can be viewed not simply as a sample of search space at one point but as a sample of the quality of numerous different subspaces or hyperplanes of the search space. Each hyperplane can be described by a schema. A schema is a template string in which some locations have fixed alleles while for other locations alleles vary over an allowable set (alphabet). For example, in binary representation 1**10*** represents a schema where locations 1, 4 and 5 are fixed while other locations are in “don’t care” condition. The number of fixed locations in a schema is called its order and the distance between its two farthest fixed locations is called its defining length. The schema described above has order 3 and defining length 4. A schema is said to match a chromosome (and conversely the chromosome is

an instance of the schema) if the chromosome matches the schema at its fixed locations. For example, both the chromosomes 11110000 and 10110010 match the schema described above. In fact a chromosome of length L matches 2^L different schemas. For example, chromosome 10 matches four different schemas **, *0, 1* and 10. Therefore evaluating the fitness of a chromosome, implicitly gives a sample of the fitness of all the schemas of which the chromosome is an instance. The schema theorem of Holland (1975) suggested that, through the application of biased selection, recombination and mutation, the short, low order schemas whose average fitness remains above the mean will receive exponentially increasing number of trials in subsequent generations. Goldberg (1989) named short, low order, highly fit schemas as building blocks and proposed the building block hypothesis, which says that the genetic algorithm seeks near optimal performance through juxtaposition of building blocks.

Despite of the fact that schema theorem and building block hypothesis have several drawbacks (Grafenstette, 1993), and many alternative theories (Altenverg, 1994; De Jong et al., 1994; Goldberg and Segrest, 1987; Horn, 1993; Nix and Vose, 1992) have been proposed to explain the behaviour of genetic algorithm, these two are still most widely used to explain the behaviour of genetic algorithm.

1.5.2 Representation of Individuals

The most important part in designing a genetic algorithm for a problem is to define the genotype and a suitable evaluation function. A genotype should be a natural representation of the phenotype in such a way that notion of distance in problem space should be preserved in representation space. The encoding should be such that all the solutions must have a representation and that each possible string corresponds to some solution. There should be minimum redundancy in representation and the evalua-

tion function should be unambiguous. Ideally a phenotype must correspond to one and only one genotype in the representation space. When more that one representation corresponds to a phenotype it severely affects the performance of the genetic algorithm (Schaffer et al., 1992).

Most commonly used representations in simple genetic algorithm are binary, integer and real-valued representations. However, there are many more representations, useful for different problems and designed specifically for them so that the information relevant to the problem should be contained in the genotype. Two such representations (permutation and grouping) will be discussed in subsequent sections.

1.5.2.1 Binary Representation

Here the genotype consists simply of a string of binary digits. The length of the bit string is decided by the parameters involved and the required precision. In the simplest case the string length is fixed. It is the simplest and initially most used representation. It is also easy to design evolutionary operators for this representation. However, in some cases the representation is not a natural one. For example, representing integers or real numbers in binary strings is not natural and better results can be obtained by using the integer or real valued representations directly.

1.5.2.2 Integer Representation

This representation is suitable where genes can take discrete values. These values may come from a set which may be either finite-valued or unrestricted. The attributes used may be ordinal or cardinal. When ordinal attributes are used, the natural relationship between the possible values of a phene must be considered while designing genetic operators.

1.5.2.3 Real-Valued Representation

When the alleles are continuous values rather than discrete then we represent these values as floating-point numbers and design genetic operators to take advantage of this representation.

1.5.3 Mutation

Mutation is used to maintain diversity in the population so that premature convergence may not take place. Moreover, to a certain extent, it also helps to explore new areas of the search space. For different type of representations different mutation operators have been proposed. All mutation operators generate a new, but random, search point without any positional bias in the neighborhood of present search point.

1.5.3.1 Bitwise Mutation

This mutation operator is for binary representation. Each bit is considered separately and allowed to flip with a probability known as mutation rate. Figure 1.1 shows the bitwise mutation operator.

Fig. 1.1: Bitwise mutation for binary representation

1.5.3.2 Random Reset Mutation

This mutation operator is used for integer representations and is an extension of bitwise mutation operator for binary representation. Here with probability defined by the mutation rate, each gene is set to a new value, chosen at random from a set of permissible values. This type of mutation operator is most suitable where gene encodes cardinal attributes. Figure 1.2 shows such a mutation for the case where possible alleles are *A, B, C,* and *D* for all the genes.

Fig. 1.2: Random reset mutation for integer representation

1.5.3.3 Creep Mutation

This scheme was also designed for integer representation and is most suitable for ordinal attributes. In this mutation we change the value of the gene to be mutated by small value rather than large ones. For instance, if in the previous examples the attributes were ordinal, then the probability of the third gene to mutate to value *C* or *A* is more than its probability to mutate to *D*.

1.5.3.4 Uniform Mutation

This mutation is used in real-valued chromosomes. In this scheme, with probability defined by mutation rate the value of each gene is changed randomly within a lower bound (L_i) and upper bound (U_i). Figure 1.3 illustrates the uniform mutation where value of each gene lies in the range [1, 8].

Fig. 1.3: Uniform mutation

1.5.3.5 Non-Uniform Mutation with a Fixed Distribution

This mutation is also for real-valued representations and is similar to creep mutation for integer representation. This mutation is implemented by adding a value (an amount drawn form a specific probability distribution) to each gene. Normally Gaussian and Cauchy distributions are used. Mean and standard deviation of the distribution are the parameters for this mutation. Normally mutation rate is taken to be 1 per gene in this mutation.

1.5.4 Recombination

The recombination is based on the principle that by mating two individuals with different but desirable features, we can produce an offspring that have desirable features of both of its parents. However, the result can be counterproductive also. But when applied many times, there must be some positive outcomes. The recombination or crossover operator should be designed in such a way that it should recombine the information relevant to the problem in hand while recombining genes. In addition, if the mating parents are genetically close then the offspring must also be similar to the parents. This is called similarity requirement. Therefore, for different representation schemes, different recombination operators are proposed.

1.5.4.1 1-Point Crossover

The 1-Point Crossover was the original crossover operator proposed in Holland (1975). A locus (crossover site) is chosen randomly and the substrings after that locus are swapped between the two parents to produce two offspring. It can be used in any representation but is generally used in binary or integer representation. This crossover, however, has positional bias. If two genes are at opposite end of a parent, those will always be separated even if they are forming a good schema. Figure 1.4 illustrates 1-point crossover operator.

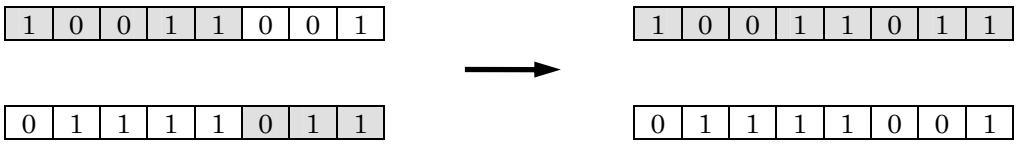

Fig. 1.4: 1-point crossover

1.5.4.2 N-Point Crossover

It is just like 1-point crossover; the only difference is that *N* random locations are chosen as crossover sites, instead of a single one. Like 1-point crossover it also shows positional bias. When *N* is odd the genes at oppo-

site ends will never be together after crossover while if N is even they will always be together. Figure 1.5 illustrates N-Point crossover for $N = 2$.

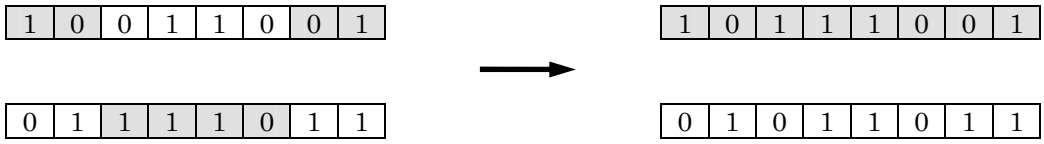

Fig. 1.5: 2-point crossover

1.5.4.3 Uniform Crossover

In this crossover each locus is treated independently. For each locus a uniform random number between 0 and 1 is generated and if it is less than a predefined value (usually 0.5), then the corresponding gene for the offspring will be taken from one parent otherwise from the other parent. However, it has a drawback that co-adapted genes from one parent may not be transmitted to the offspring. This crossover can also be used with any representation. Figure 1.6 illustrates uniform crossover with the help of an example.

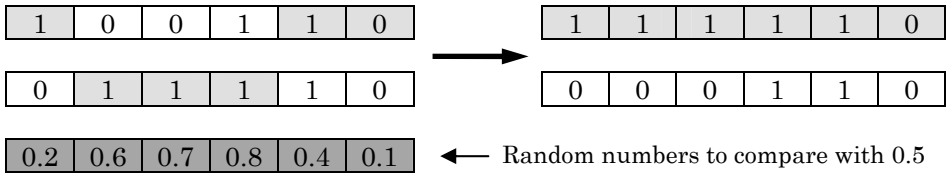

Fig. 1.6: Uniform crossover

1.5.4.4 Arithmetic crossover

In real-valued representation simple recombination is usually not much useful. Here another recombination scheme called arithmetic crossover is used. In arithmetic crossover, a part of the offspring or the entire offspring is produced by numerical interpolation of the corresponding loci of the two parents. If r is a uniform random number in the range (0, 1), the expression for the offspring in terms of the parents may be given as

$$X' = rX + (1 - r)Y$$

$$Y' = (1 - r)X + rY$$

where X and Y are two parents and X' and Y' are two offspring. If instead of complete offspring only a part of the offspring is produced as mentioned above, the remaining part of the offspring comes from the corresponding loci of one or the other parent. In this way, recombination is now able to create new genetic material and also, in a sense uses knowledge of past exploration. Therefore, the genetic algorithm does not have to depend only on mutation for generating new genetic material. So this crossover operator complements mutation in some sense. Figure 1.7 illustrates the arithmetic crossover operator with the help of an example where the interpolation with $r = 0.2$ is applied on loci 2, 3, 4 and the remaining loci are copied from one of the two parents.

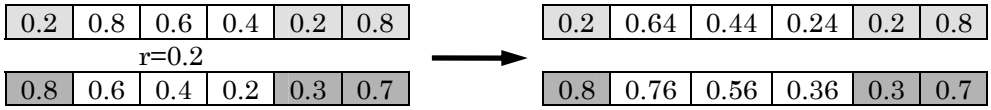

Fig. 1.7: Arithmetic Crossover

1.5.5 Population Models

The composition of next generation in terms of individuals origin is decided by population model. There are mainly two versions of genetic algorithm in this respect: The generational genetic algorithm and the steady-state genetic algorithm.

1.5.5.1 Generational Genetic Algorithm

In generational model after each generation the whole population is replaced by its offspring.

1.5.5.2 Steady-State Genetic Algorithm

In steady-state genetic algorithm a certain number of old individuals are replaced by same number of new offspring. The fraction λ/μ, where μ is the population size and λ is the population that is replaced, is called the generational gap (Whitley and Kauth, 1988; Davis, 1991). Usually $\lambda = 1$ is taken, implying that each time two parents are selected to produce a single offspring that replaces a less fit member of the population. The steady-state population model has an advantage over generational model since the best solutions are always kept in the population and the child is immediately available for selection and reproduction. Thus we can possibly find better solutions faster. Another advantage lies in the ease with which we can avoid duplicate copies of the same individual in the population. In the generational approach multiple copies of the same individual may exist in the population. Normally these individuals are among the best individuals but they can quickly dominate the whole population. In this situation, no further improvement is possible without mutation and a much higher mutation rate is required to get further improvements. In the steady-state approach the child can be checked against existing population members and if it is identical to any existing individual in the population then it is discarded. In this way duplicate solutions are avoided and the problem of premature convergence is averted. Steady-state population model is the most widely used population model nowadays.

1.5.6 Selection Methods

The selection methods basically differ in selection pressure and degree of randomness by which parents are selected. Three types of selection methods are normally used.

1.5.6.1 Fitness Proportionate Selection

In this method the probability of selection of an individual depends on the absolute fitness value of the individual compared to that of rest of the population (Holland, 1975). This probability determines how many copies of each individual will go into the mating pool. Obviously the number of copies calculated is normally not an integer. To map these numbers to integers so that their sum is a predetermined integer (either equal to population size or some fraction of it), different sampling methods are used. Most widely used of them are roulette wheel algorithm (Goldberg, 1989) and stochastic universal sampling (Baker, 1987).

The fitness proportionate selection however have following shortcomings in different situations –

- Premature convergence due to outstanding individuals. During initial generations, fitness variance in the population is high and a few individuals are much more fit than others. Under fitness proportionate selection scheme, these individuals and their descendent will take over the population very quickly and hence preventing the genetic algorithm from doing any further exploration of the search space.
- Low selection pressure due to small separation in fitness values and hence slow performance of the genetic algorithm. This problem normally occurs during later generations of genetic algorithm.

To avoid these problems in fitness proportionate selection, various scaling methods are used (Mitchell, 1996).

1.5.6.2 Ranking Selection

To address the shortcomings of fitness proportionate selection, rank based selection method was proposed (Baker, 1987). In this method, the individuals are allocated ranks according to their relative fitness. The selec-

tion probability is decided on the basis of this rank using some mapping function so that better individual may get more chance.

Selection pressure may be changed by changing the mapping function or some control parameter in it. For example, linear mapping has less selection pressure as compared to exponential mapping.

1.5.6.3 Tournament Selection

The previous two selection methods require knowledge about the entire population. However, there are situations where the population size is too large or the population is distributed in some way (for example, in parallel implementation of genetic algorithm). Similarly there are situations where there can be no global definition of fitness. In such cases it is either impossible or very difficult to obtain information about the fitness of each member of the population. Tournament selection method is designed in such a way that it does not require any global knowledge of the population. Its simplest and most commonly used form is binary tournament selection where two individuals are selected at random and better of the two is copied to the mating pool with probability p, otherwise the worse is copied. In k-ary tournament selection, k individuals instead of two take part in selection.

The binary tournament selection has the same effect as ranking selection but in general, tournament selection suffers from the same problems as the roulette wheel algorithm (Goldberg and Deb, 1991). However due to its extreme simplicity and the fact that the selection pressure can be controlled easily by varying the tournament size, this method is most widely used.

1.5.7 Survivor Selection

The methods that decide which offspring together with individuals of current generation will go to next generation are called survivor selection

methods or population replacement schemes. Commonly used population replacement schemes are (i) age-based replacement (ii) fitness-based replacement. In age based replacement, each individual exists in the population for a fixed number of genetic algorithm iterations while in fitness-based replacement either worst members of the population are replaced or few best elites in the population (elitism) are always preserved.

1.6 Genetic Algorithm for Permutation Problems

There are problems such as "Job Shop Scheduling" or "Travelling Salesman" in which the alphabetical labels or numerical values of objects are not important. Rather it is the adjacency or the ordering of objects which is of prime importance. Therefore none of the representation schemes discussed so far are suitable for these problems. These representations are not only inefficient (because of redundancy), but there is also a large possibility of invalid constructions as a result of genetic operations. Therefore these problems require different type of representation and genetic operators for them. Permutation representation is the most natural representation for such problems. This representation allows different permutations of a set of alphabets as genotypes for different solutions. For the permutations to be valid, each possible allele must occur exactly once in the chromosome. Therefore any string in which an alphabet occurs twice or more, or an alphabet is missing represents an invalid permutation and hence is ruled out. To preserve the permutation property, appropriate mutation and recombination operators must be designed. These operators must propagate exactly the genetic information relevant for the problem to the offspring. However, a fair amount of redundancy may still be present, i.e., there may exist more than one permutation representation of a solution.

1.6.1 Mutation

If we consider each gene separately in the permutation representation and try to mutate it with any possible mutation method for simple genetic algorithm, we will get an invalid permutation. To overcome this problem a number of specialized mutation operators have been proposed that preserve permutation property.

1.6.1.1 Swap Mutation

In this type of mutation, alleles at two different loci are swapped therefore mutating the order as well as adjacency information. Figure 1.8 illustrates swap mutation.

Fig. 1.8: Swap Mutation

1.6.1.2 Insert Mutation

An allele at one locus is made to follow or precede another locus while shifting rest of the string right or left accordingly. Figure 1.9 illustrates insert mutation.

Fig. 1.9: Insert Mutation

1.6.1.3 Scramble Mutation

A substring is chosen and alleles inside it are scrambled to generate new adjacency and order. Figure 1.10 illustrates scramble mutation.

Fig. 1.10: Scramble Mutation

1.6.1.4 Inversion Mutation

A randomly chosen substring is inverted. It is considered to be best mutation operator for adjacency based problems like travelling salesman problem. Figure 1.11 illustrates inversion mutation.

Fig. 1.11: Inversion Mutation

1.6.2 Recombination

A number of specialized recombination operators have been proposed for permutation representation, keeping in mind the desired characteristics of recombination operators. The most commonly used crossover operators are described below. It is to be noted that depending on the situation (for example, in steady-state genetic algorithm) each of these crossover operators can be used to generate only a single offspring.

1.6.2.1 Partially Matched Crossover (PMX)

It was proposed by Goldberg and Lingle (1985) for the Travelling Salesman Problem. The PMX has become one of the most widely used crossover operator for adjacency type permutation problems. In PMX, two crossover sites are chosen uniformly at random. PMX proceeds by swapping the positions of those alleles, in each parent separately, which are at the same position in the two parents between the crossover sites. Each offspring contains ordering information partially determined by each of its parents. In the example given in Figure 1.12 the positions of 6 and 8, 1 and 4, and, 5 and 1 are swapped in both the strings separately as the location of the former allele in each pair in the first parent is same as that of later allele in the second parent. This crossover however has shortcoming that it fails to preserve the adjacency information common in both the parents.

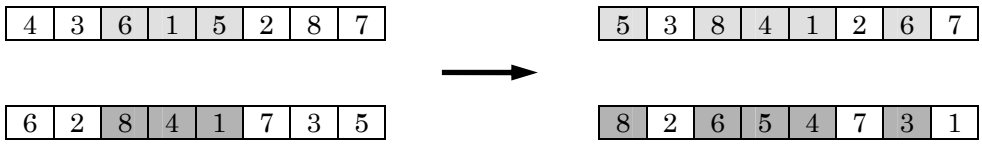

Fig. 1.12: The PMX

1.6.2.3 Order Crossover

The order crossover was proposed by Davis (1991) for order-based circular permutation problems. In this operation like PMX, two crossover sites are chosen uniformly at random. Now the substring between these sites in the first parent goes to the first offspring and the corresponding substring in the second parent goes to the second offspring at the same location. Starting from the second crossover site, rest of the alleles will be copied in each offspring in the order in which they are in the other parent treating chromosome as circular. Figure 1.13 illustrates the order crossover.

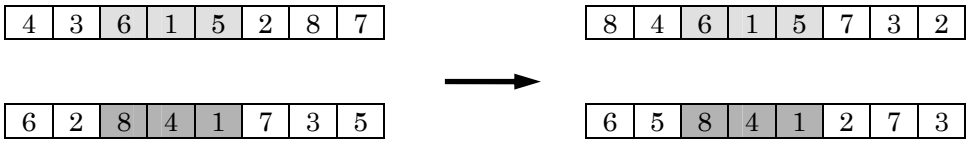

Fig. 1.13: The order crossover

1.6.2.3 Cycle Crossover

Cycle crossover (Oliver, Smith and Holland, 1987) tries to preserve the absolute positions of alleles as much as possible. The operator divides the alleles into cycles in both the parents. A cycle is a subset of alleles in which each allele always occurs paired with another of the same cycle when the two parents are aligned. The offspring are created by choosing alternate cycles from each parent. The procedure for detecting a cycle and constructing a partial child from the elements of the cycle is as follows:

(i) The first unused locus α of first parent is selected.

(ii) The allele at locus α of the first parent is inserted to the child at the same locus.

(iii) The allele in the second parent at the locus α is noted and its location in the first parent is traced. Now α is reset to this location.

(iv) Repeat steps (ii) and (iii) until the allele which was chosen foremost in this cycle is repeated.

The roles of the two parents are interchanged each time a new cycle is found. Figure 1.14 explains the cycle crossover with the help of an example where there are three cycles comprising alleles 1, 3, 8; 2, 4, 7; and 5, 6 respectively. The first child is made of first and third cycle from the first parent and second cycle from the second parent. Likewise, the second child takes first and third cycle from the second parent and second cycle from the first parent.

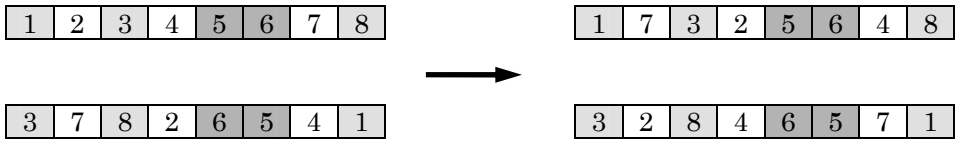

Fig. 1.14: The cycle crossover

1.6.2.4 Uniform Order Based Crossover

The uniform order based (UOB) crossover was proposed by Davis (1991) for the non-circular permutation problems where relative ordering between objects is important, rather than their absolute ordering. UOB crossover takes two parents and tries to copy allele at each position of one of the parents with probability 0.5 to the first offspring. The remaining positions in the first offspring are filled from the unused alleles in the order in which they appear in the second parent. The second offspring is created in an analogous manner by interchanging the role of the two parents. Figure 1.15 illustrates the UOB crossover.

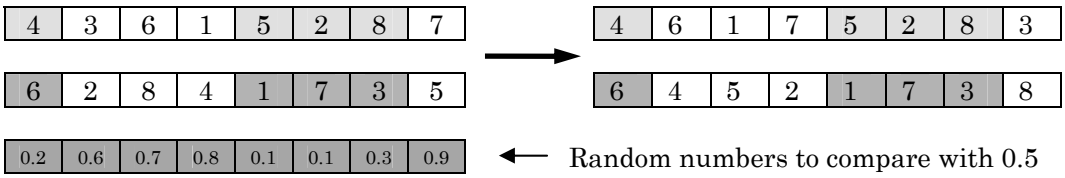

Fig. 1.15: The UOB crossover

1.7 Grouping Genetic Algorithm

As for the ordering and adjacency problems permutation representation is required, in a similar manner for grouping problems such as bin packing, task allocation, we need a different representation. The representation schemes discussed so far are not suitable for the grouping problem because of the following reasons (Falkenauer, 1998):

- These representations are highly redundant when used to encode a grouping problem. For the same phenotype many different genotypes are possible.
- For grouping problems, the genetic operators discussed so far are context insensitive and too much disruptive.

In fact two types of representation were proposed for grouping problems initially. In the first scheme known as positional scheme, locus specifies the object while the corresponding allele specifies the group to which this object belongs, for example, *ABAC* represents that first object and third object belongs to group *A*, while second and fourth objects belongs to group *B* and *C* respectively. In positional scheme, it is the composition of each group that is important rather than group name. For example, *ABAC* and *CBCA* represent the same solution. Such a scheme yields constant length chromosome, thus allowing the application of standard genetic operators.

The second scheme was permutation-based in which a chromosome is represented as a permutation of objects. A decoder is needed to convert the

chromosome into phenotype, i.e., decoder gives the actual distribution of objects into groups and the value of the cost function. This scheme also yields constant length chromosomes.

In both the above schemes, any allele occupying a locus has sense only in the context of that particular string. Replacing it with any other allele or swapping it with any other allele at any other locus will bring it out of context. The outcome may be invalid or at least it may not fulfil the similarity requirement which says that two identical parents should produce a similar child. For example, the individuals *ABCADD* and *BADBCC* represent the same solution therefore when recombined they should produce a similar progeny. The two point crossover of the above individuals may produce a child *AADADD* which is obviously not similar to their parents. In many cases this construction may be invalid even. Moreover the disruption caused by the operation is obvious. The child is comprised of only two groups in comparison to four in its parents. Recombination and mutation operator for permutation encoding scheme also suffer from similar disadvantages. For example, consider the permutation 12345678. The swap mutation between loci 1 and 8 generates offspring 82345671. Depending on the problem, the two permutations may represent quite different solutions.

The explanation of the above disadvantages lies in the fact that none of these schemes use grouping information as building blocks which is a vital requirement for solving the grouping problems. Therefore Falkenauer (1993) proposed a different scheme which uses grouping information as building block. He called his scheme the grouping genetic algorithm (GGA).

1.7.1 Representation in Grouping Genetic Algorithm

In the GGA, the chromosome has two parts: first part is called the standard object part and the other is called the group part. The standard ob-

ject part shows which object belongs to which group, while group part just represents groups present in the standard object part on one gene per group basis. For example, in *AFBDBCFF* : *FCBDA*, the first part, i.e., the standard object part represents that first object belong to group *A*, second to group *F*, third to group *B* and so on. The second part, i.e., the group part represents that there are 5 groups viz. *A*, *B*, *C*, *D*, *F* in the chromosome. Similarly *CCDDDCCC* : *CD* represents a solution with just two groups. Genetic operators especially crossover for the GGA are designed to work on the group part only. As the number of groups may vary from one individual to another the operators have to deal with variable length chromosomes.

1.7.2 Mutation

Almost all mutation operators for the grouping problems are based on one of the following approaches (Falkenauer, 1998):

- Eliminate some groups at random from the solution and then reinsert objects belonging to these groups into the solution using some problem specific heuristic.
- Remove some objects at random from their respective groups and then reinsert these objects into the solution using some problem specific heuristic.
- Create a new group using some objects selected randomly or according to a specific criterion and delete these objects from their original groups. If some groups become empty during the process then remove these groups from the solution.

1.7.3 Recombination

The recombination operator essentially operates on the group part. In GGA the notion of locus becomes fuzzy. When parts are aligned, two groups may or may not overlap. Another possibility is that one group may

be a proper subset of another group. In the former two cases, loci may be comfortably said to be equal or unequal, but in the last case they are neither equal nor unequal but something in between. In addition to this too long a chromosome will naturally be a poor one and there will be a limit on the length of the chromosome decided by the number of objects. Keeping in mind these facts, the straightforward crossover at grouping part is not possible.

Crossover operator of Falkenauer (1998) for the grouping problems consists of inserting the groups between the two randomly chosen crossover sites of one parent at the first crossover site of the second parent. Now there will be some objects which appear in two groups. These objects will be deleted from the groups originally belonging to the second parent. Depending on the problem, next step may consist of adapting the resulting groups according to the hard constraints of the problem and cost function to optimize. In many grouping problems this step consist of eliminating the groups which are modified by the insertion. The objects belonging to these groups will be reinserted into the solution using some problem specific heuristic. The second child is generated in a similar way by interchanging the role of two parents.

Another possible approach for crossover is to iteratively copy the most suitable (according to the problem under consideration) remaining group from one of the two parents to the child (Galinier and Hao, 1999; Greene, 2001). This approach will be discussed in detail in chapter 7.

1.8 Outline of the Thesis

This thesis is focussed primarily on solving some NP-Hard graph and grouping optimization problems through genetic algorithm. It is divided into eight chapters beginning with this introduction. Chapters 2 to 6 deal

with graph problems whereas last two chapters are devoted to two grouping problems. In the following, we outline the content of each of these chapters.

Chapter 2 deals with the maximum clique problem. In this chapter a hybrid approach combining a genetic algorithm and a local search heuristic has been proposed for the maximum clique problem. Computational results on standard benchmark instances show that the proposed approach outperforms all the other evolutionary approaches for the maximum clique problem. However the obtained results are much inferior to those obtained with the best non-evolutionary heuristic for the maximum clique problem

Chapter 3 extends the approach developed in chapter 2 to the maximum weight clique problem. The extended approach outperforms the two best heuristics for the maximum weight clique problem.

Chapter 4 extends the approach developed in previous two chapters to the minimum weight vertex cover problem. The resulting approach outperforms a recently proposed Ant-Colony Optimization based method for the minimum weight vertex cover problem.

In Chapter 5, a perturbation based local search is developed for the degree-constrained minimum spanning-tree (DCMST) problem. In this chapter we propose the concept of domains as a generalization of the concept of neighborhoods of a solution and a variable-domain-descent local search as a generalization of variable-neighborhood-descent local search and demonstrate their effectiveness in solving the DCMST problem. Our perturbation based local search not only outperforms all other methods for the DCMST problem but also finds new best solution values for the three problem instances.

In Chapter 6 some improvements are suggested to an already existing greedy heuristic and permutation-coded evolutionary algorithm for the

bounded-diameter minimum spanning-tree (BDMST) problem. The suggested improvements not only increase the solution quality but also dramatically reduce the running time.

Chapter 7 deals with one-dimensional bin-packing problem. In this chapter first a new hybrid grouping genetic algorithm is proposed for the problem. Next some modifications are suggested to an already existing heuristic for the one-dimensional bin-packing problem. Finally we present a combined approach using the hybrid grouping genetic algorithm and the modified heuristic. The performance of the combined approach is comparable to the best method known so far for the problem. The combined approach also finds new best solution values for the three problem instances.

In chapter 8, we propose two different approaches for the equal piles problem. The first approach extends the hybrid grouping genetic algorithm developed in chapter 7 to the equal piles problem. The other is a perturbation based local search. We also introduce some new benchmark instances for the problem. The computational results show the effectiveness of our approaches.

Among the seven problems attempted in this thesis, the maximum clique and the one-dimensional bin-packing problems are two most widely studied problems, whereas the equal piles problem is a relatively new and not so well studied problem.