15

Solving Optimization Problems in Wireless Networks Using Genetic Algorithms

15.1 Introduction

The surging demand for mobile communications and the emerging field of ubiquitous computing require enormous development in the realm of wireless networking. Although many problems in mobile wireless networking can be successfully solved using techniques borrowed from wireline networks, there exists some problems that are very specific to the wireless domain and often computationally very difficult or sometimes even intractable. This is primarily due to the inherent limitations of wireless communications, such as scarce bandwidth, high bit error rate, or location uncertainty. Most of these problems can be mapped to classical optimization problems, with the goal of optimizing some objective functions, say resource utilization, while satisfying all the constraints imposed by the wireless communication systems. The constraints make most of the problems NP-complete or NP-hard [1]. There are two broad approaches to solve such problems. One is to directly compute the exact solution based on the constraints, using a brute force technique. However, this approach is often infeasible in a large-scale problem domain. The other approach is to use a heuristic-based solution that can be computed in feasible time. Although a heuristic may not yield the optimal solution, a carefully designed heuristic may produce a near-optimal solution with low computational complexity. Various attempts have been made to develop efficient heuristics that range from calculus-based search methods to random search techniques. The paradigm of "evolutionary
computing or programming” essentially provides such heuristics for solving computationally difficult problems, by mimicking the evolution process that the nature uses to eliminate weak forms of life.

An important example of evolutionary computing is genetic algorithm (GA), which is a guided, random search technique. In GA, the less fit (or incompetent) solutions are replaced by better solutions produced by applying some genetic operators on the existing ones in the solution space. This evolutionary learning process progressively refines the search space and makes the problem computationally manageable, thus enabling the search procedure to converge quickly and resulting in a near-optimal solution in feasible time. GA has been successfully applied to solve several optimization problems in the wireless domain [2–4]. The common objective of such problems is the optimal usage of scarce and hence costly wireless resources, such as bandwidth. For example, in References 3 and 5, GA has been used to find an optimal location management strategy for cellular networks. The goal here is to determine an optimal policy for location updates of a mobile host, which ensures minimal paging and signaling overhead, thereby conserving the costly wireless bandwidth. Whereas this work considers only a single optimization criteria (the location management cost), a multi-objective GA-based location management framework has been proposed in Reference 6 that considers other optimization factors, such as the load balancing between various mobility routers in an Universal Mobile Telecommunication Systems (UMTS) [7] network. Channel assignment is another challenging problem in cellular networks. The entire spectrum of wireless bandwidth is divided into several channels and assigned to the cells, with the constraint that no two adjacent cells are assigned the same channel, which ensures that there is no co-channel interference. A single cell may have more than one channels. The channel assignment problem can be shown to be equivalent to the graph coloring problem, and hence NP-hard [1]. GA-based channel assignment algorithms [2] provide a scalable mechanism to determine the least number of channels required for a particular cellular network, while satisfying the constraint related to the co-channel interference problem in adjacent cells. Due to the limited resource available in wireless networks, the admission of new calls must be monitored to ensure a minimum quality of service (QoS) for the already admitted calls. This is done by call admission control algorithms, which essentially select, from a large number of possible admission policies, the optimal one with overall service quality improvement and minimum call blocking rate. However, the number of possible policies can be very large such that finding the optimal policy can be computationally intractable. Again, GA provides a heuristic-based approach [4] to yield a near-optimal call admission policy that ensures QoS for the admitted calls while minimizing the call blocking rate. GA has also been used in solving QoS routing problems in networks. It can be shown that unicast routing with two or more QoS constraints can be mapped to a NP-hard problem. Naturally, the problem is also NP-hard for multicast routing. The imprecise network state information in dynamic wireless networks further complicates the problem. In Reference 8, GA has been used to produce near-optimal solutions in computationally feasible time, for multicast routing problem in networks with imprecise state information, while satisfying more than one QoS constraints.

The goal of this chapter is to present a review of GA-based solutions of the above problems. The rest of this chapter is organized as follows. Section 15.2 reviews the basic concept of GAs and their interpretation. The location management problem is discussed in Section 15.3. Section 15.4 presents a GA-based solution for optimal channel allocation in cellular networks. Efficient call admission control in cellular networks is described in Section 15.5, while the design and performance analysis of an efficient multicast routing protocol is described in Section 15.6. Finally, Section 15.7 concludes the chapter.

15.2 Basics of Genetic Algorithms

GAs are stochastic algorithms, whose search methods model a natural phenomenon, genetic evolution. In evolution, the problem each species faces is one of searching for beneficial adaptations to complex changing environments. The “knowledge” that each species has gained is embodied in the makeup of the chromosomes of its members. In this light, GAs can also be classified as a learning technique. They have
been successfully applied to tackle optimization problems such as scheduling [9], adaptive control [10], game playing [11], cognitive modeling [12], transportation problems [13], traveling salesman problems [14], database query optimization [15], and so on. Although GAs belong to the class of probabilistic algorithms, they differ from random search algorithms because they combine elements of directed and stochastic search. Therefore, GAs are also more robust than purely directed search techniques [16].

As an evolutionary programming model, a GA for a particular problem must have the following five properties:

- A genetic representation of the potential solution space to a symbolic domain, such as a strings of symbols, which form the so-called *chromosomes*.
- A methodology to create an initial population of potential solutions.
- An evolution function that plays the role of the environment, rating solutions in terms of their “fitness.”
- Genetic operators that alter the structure of chromosomes.
- Values for various parameters such as population size, probabilities of applying genetic operators, etc.

Figure 15.1 shows a pseudo-code depicting the generic framework of any GA. An initial random population is created and each member of the population is evaluated using a fitness function. Three major genetic operators, selection, crossover, and mutation, are then successively applied to the members of the population in each generation until the stopping criteria is met or there is little change in the quality of the best solution in the entire population. *Selection* is the operation that increases the number of good solutions or chromosomes in a population; usually the best chromosome in each phase of evolution is selected. The *crossover* operation generates new representative chromosomes containing the properties of more than two chromosomes. Finally, *mutation* changes or mutates a portion of a chromosome to introduce a new chromosome in the population. When there is little or no change in the quality (or fitness) of the chromosomes in the population with successive application of GA operators, the best one is selected to represent the final solution.

15.3 Location Management

The location management problem is defined as the tracking of a mobile user in cellular wireless networks. As shown in Figure 15.2, a cellular network is divided into discrete cells, each served by a base station (BS). Mobile users communicate with a BS using wireless links. A number of such BSs are linked to a mobile switching center (MSC), which is connected to the existing wireline networks. The location management problem has two flavors: one on the part of the mobile device and the other on the part of the network system. In a stand-by mode, every mobile device is made to report its location in regular intervals. This process is known as *registration or update*. On the arrival of a new call to the mobile device, the network system initiates a search for the device by polling the cells, where it could be possibly
located. This procedure is known as paging. By giving a low upper bound on the maximum number of cells that can be polled, the paging process reduces the paging cost, but requires more frequent updates, thus increasing the update cost. On the other hand, a reduction of number of updates essentially decreases the update cost, but increases the location uncertainty and the subsequent paging cost. The essence of an optimal location tracking thus lies in minimizing the combined update and paging costs associated with a particular mobile device. The concept of using a suitable GA-based approach to minimize the average per-user location management cost was first proposed in Reference 3.

A cellular network can be represented by a graph $G(V, E)$, where a node represents the location areas (LA — a logical representation of one or more cells) and an edge represents the access paths between a pair of location areas. At this point, the most interesting question is whether a mobile device will issue an update or not upon entering a new LA. Let $\delta_i$ represent an update decision variable for a user in LA$_i$(1 $\leq i \leq M$), such that $\delta_i = 1$ or 0 depending on whether or not an update has been issued. Now, if Cost$_p(i)$ and Cost$_u$ respectively denote the cost associated with paging LA$_i$ and issuing a single update, then the total location management cost (LMC) is determined by taking a weighted average of the LMCs in the individual LAs. This is mathematically represented by: $\text{LMC} = \sum_{i=1}^{M} \frac{\Pi_i}{\lambda^{\pi_i}} \times \text{LM}_i$, where $\Pi_i$ is the normalized weight associated with the LMC, $\text{LM}_i$, of LA$_i$. The average LMCs for both cases ($\delta_i = 1$ and $\delta_i = 0$) depend on the individual update cost, paging cost, call arrival rate, and the user’s residence time in the LA. Assuming a Poisson arrival with rate $\lambda$, geometrically distributed (with parameter $p_i$) residence time, it has been shown in Reference 3 that $\text{LM}_i = \text{Cost}_u + \lambda / p_i$. It is now clear that an update strategy $U_i = [\delta_i]$ for the user constitutes a vector of decision variables, having values 0 or 1 for all the LAs. The objective is to obtain optimal strategy $U^*_i$ such that the LMC is minimized. While enumerating all possible update strategies, the state space of the solution increases exponentially with the number of LAs. Therefore, a GA is proposed in Reference 3 to obtain a (near-) optimal solution for the location management problem.

As mentioned in Section 15.2, the first step in a GA-based approach is to map the state space into a symbolic domain. The most obvious way is to represent each bit-string associated with a strategy $U_i$ by a single chromosome (or genome). The length of every chromosome is equal to the number of LAs. A group of strategies is chosen to form the initial population. For faster convergence, the relative proportion of
the number of 0s and 1s in the bit-string is chosen based on the call arrival rate and update cost. For a relatively low call arrival rate and high update costs, it is wise to issue less frequent updates, thereby resulting in more 0s than 1s in the chromosomes. Since, at every iteration, the GA inherently attempts to increase the associated fitness values, the fitness function is chosen to be reciprocal of the total LMC, that is, 1/LMC. The roulette wheel spinning selection [16] is used with elitism so that better chromosomes will survive for the next iteration. After this selection, the crossover and mutation functions are executed with probabilities 0.8 and 0.01 respectively. The fitness of the children are now evaluated and the entire process is repeated.

**Illustrative Example:** At each iteration of the GA, the best chromosome, from the initialization phase till that iteration cycle, is tracked. This gives the optimal (or near-optimal) solution at the termination of the algorithm. The population size for each generation is kept constant at 50 and the number of bits (i.e., the number of LAs) in the chromosome is chosen as 8. The cost function LMC is computed using the steady-state transition probabilities between any two LAs. It was found that the GA converged very fast to the near-optimal solution. The population size is kept constant at 20 and in all the cases the algorithm converges to the optimal solution within 1000 generations. The best and average values of the fitness of the chromosomes as well as the standard deviations are computed for each generation. A sample run of the entire process with different generations having best and average fitness values is shown in Table 15.1, which results in 11011 as the best chromosome having fitness 0.127 corresponding to LMC = 7.87 units. Table 15.2, on the other hand, represents the optimal update strategies for different call arrival rates and update/paging cost ratios.

Recently, the location management problem is also investigated in Reference 5 using a combination of cellular automata (CA) and GAs. The total LMC is estimated as \( r \sum_{i \in C} w_{mi} + \sum_{j=0}^{N-1} w_{cj} v(j) \), where \( r \) is the update-to-paging cost ratio, \( C \) is total number of reporting cells, that is, the number of cells from which at least one update is issued, \( w_{mi} \) represents frequency of movement into a cell \( i \), \( w_{cj} \) represents frequency of call arrival within a cell \( j \), and \( v(j) \) represents the vicinity (neighborhood) of cell \( j \). Note that

<table>
<thead>
<tr>
<th>TABLE 15.1</th>
<th>Sample Run of Genetic Algorithms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generation no.</td>
<td>Best fitness</td>
</tr>
<tr>
<td>1</td>
<td>0.102</td>
</tr>
<tr>
<td>3</td>
<td>0.106</td>
</tr>
<tr>
<td>7</td>
<td>0.107</td>
</tr>
<tr>
<td>15</td>
<td>0.120</td>
</tr>
<tr>
<td>20</td>
<td>0.127</td>
</tr>
<tr>
<td>35</td>
<td>0.127</td>
</tr>
<tr>
<td>50</td>
<td>0.127</td>
</tr>
<tr>
<td>100</td>
<td>0.127</td>
</tr>
<tr>
<td>200</td>
<td>0.127</td>
</tr>
<tr>
<td>300</td>
<td>0.127</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TABLE 15.2</th>
<th>Optimal Update Strategies</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \frac{C_{ul}}{C_p} )</td>
<td>Optimal strategy</td>
</tr>
<tr>
<td>( \lambda = 0.5 )</td>
<td>( \lambda = 0.5 )</td>
</tr>
<tr>
<td>0.10</td>
<td>00000</td>
</tr>
<tr>
<td>0.33</td>
<td>00000</td>
</tr>
<tr>
<td>0.50</td>
<td>00000</td>
</tr>
<tr>
<td>0.67</td>
<td>00000</td>
</tr>
<tr>
<td>0.75</td>
<td>00000</td>
</tr>
<tr>
<td>0.90</td>
<td>00000</td>
</tr>
<tr>
<td>1.00</td>
<td>00000</td>
</tr>
</tbody>
</table>

© 2006 by Taylor & Francis Group, LLC
cellular automata is a decentralized, discrete space–time system and the state of every cell is governed by its surrounding cells. Each cellular unit of cellular automata is associated with each cell in the network. Each cell is represented either by “1” or “0” depending on whether or not it is a reporting cell. This leads to two possible states for each cellular unit in the CA. Considering the hexagonal cells of the network, the maximum neighborhood of every cell is taken as 6. For cells with less than 6 neighbors, dummy cells, represented by “2” are added. Thus, as shown in Figure 15.3, each cell itself has 2 possible states (0 or 1) and each of its 6 neighbors has 3 possible states (0, 1, or 2), implying a total number of $3^7 \times 2 = 1458$ possible neighborhood states. Hence, the corresponding rule is of length 1458 bits and there can be a total of $2^{1458}$ transition functions. Genetic algorithms are used to search for the best one from this exponential number of transition rules. In Reference 5, an initial population of 1000 rules are created with a random value for each rule. At every iteration, a new set of CA test data is generated. The set consists of four randomly generated reporting cells configuration. The fitness function is chosen as the sum of the LMCs of all these four configurations. The selection strategy is used to get the minimum fitness values (minimum location management costs). A two-point crossover [16] with probability 0.8 and a mutation with probability 0.01 is used to achieve better solutions at every iteration. A set of 80 rules (chromosomes) have been used for elitism, while crossover and mutation have been applied on the rest of the 920 rules. Simulation results on different $4 \times 4$ and $5 \times 5$ cellular networks result in a cost per call arrival between 12.25 and 16 within 200 generations.

A careful look at both of the above mentioned location management strategies reveals that the only objective of both the schemes is to reduce the overall LMC. However, more recently, the problem of location management is combined with load balancing constraints, thereby introducing the notion of multi-objective optimization techniques. Genetic algorithms can be easily extended to solve such multi-objective optimization problems. A multi-objective, hierarchical mobility management optimization for UMTS networks has been proposed in Reference 6, which balances the load among various unique logical areas in the network, such as the LA, and the mobility routers, such as the mobile switch centers (MSCs), Radio Network Controller (RNC), and Serving GPRS Node (SGSN), while minimizing the signaling cost for location update. A schema-based niched Pareto GA [17] has been used that deals with multiple objectives by incorporating the concept of Pareto domination in its selection operator, and applying a niching pressure to spread out its population along the Pareto optimal tradeoff surface. The fitness function or the cost functions used are the RA load balancing and the intra-SGSN signaling cost, which covers the intra-SGSN routing area update and paging cost.

### 15.4 Channel Assignment

A major challenge associated with cellular wireless networks is the limitation on available bandwidth. In a time division multiple access (TDMA) scheme, the wireless bandwidth is divided into different channels
and each cell is allocated to a certain number of channels depending on its traffic density. In the cellular system, the same frequency cannot be used in the adjacent cells, as there will be co-channel interference. The hexagonal cell structure creates a cluster of 7 cells, where the frequencies will differ from each other. Thus, channels used by a cell can be reused by a different cell belonging to a different cluster (sufficiently far apart), so that the interference is bounded by some specific value. This is demonstrated in Figure 15.4, where $F_i$ represents the different frequencies used in the corresponding cell.

The channel assignment technique [2] can be static or dynamic. In a static assignment, a fixed number of channels are allocated to each cell and are estimated by the traffic demand in that cell. On the other hand, in a dynamic allocation strategy, the channels are assigned on demand, and no cell has exclusive control on any channel. Channel assignment strategies need to tackle two types of channel interferences: (i) co-channel interference and (ii) adjacent channel interference. Co-channel interference occurs when a different signal is received in the same channel as the original signal and therefore cannot be eliminated by the receiver itself. Adjacent channel interference, on the other hand, is caused by inadequate/incomplete filtering of unwanted modulation products in frequency modulation systems, improper tuning, or poor frequency control in either the reference channel or the interfering channel, or both. The channel assignment strategy needs to maintain both the co-channel and the adjacent channel interference below a tolerable limit. Most of the research in this area is focussed on finding the minimum bandwidth spectrum to satisfy a given traffic load, while maintaining the interference constraint.

Let the available frequency spectrum consist of $M$ consecutive channels $f_1, f_2, \ldots, f_M$. Also, let $N$, $d_i$, and $c_{ij}$ respectively represent the number of cells, channel demand of cell $i$, and frequency separation needed between any two channels allotted to a pair of cells $i$ and $j$. The matrices formed by the elements $c_{ij}$ and $d_i$, $(1 \leq i, j \leq N)$, are denoted as $C$ and $D$, respectively. Intuitively, $c_{ij} = 1$ and $c_{ij} = 2$, respectively mean that the same and adjacent frequencies cannot be assigned to certain channels. The channel assignment strategy now needs to find out a suitable frequency assignment matrix $F = [f_{mi}]_{M \times N}$, where $f_{mi}$ is binary-valued, with 1 indicating that the frequency $f_m$ is assigned to the $i$th cell and 0 indicating that it is not. The problem now boils down to minimize the value of $M$ such that the channels can be allocated to all the cells without interference, that is, when $f_{mi} = f_{j\hat{}} = 1$, $|f_{mi} - f_{j\hat{}}| \geq c_{ij}$ and $\sum_{m=1}^{M} f_{mi} = d_i$. It has been shown in Reference 18 that the channel assignment problem is equivalent to the graph coloring problem (NP-hard) [1], even when only the co-channel constraints are considered, that is, when the matrix $C$ is binary-valued. The actual channel assignment problem considered here is even more complex and the solution space increases exponentially with the increasing number of cells. The role of evolutionary algorithm now comes into play to obtain a (near-) optimal solution [2] in a polynomial time.
The first step to solve this problem using GA is again to generate the initial pool of encoded strings or chromosomes. Let, \( S_1, S_2, \ldots, S_P \) represent the \( P \) strings of an initial population. Each \( S_i \) is an \( M \times N \) matrix, where \( M \) and \( N \) are number of frequencies and number of cells, respectively. The elements of the matrix \( S_i \) can have values 0, 1, \(-1\), or 9. The interpretation of these four different values are enumerated here:

1. 0: Cell (given by the column number) is not using the frequency (row number), and even if it uses that frequency, there will be no conflict with other existing allocations.
2. 1: Cell is using the particular frequency.
3. \(-1\): Cell is not using the frequency (row number), and cannot use that frequency for possible interference.
4. 9: It is used at the head of all unused channels (rows).

**Illustrative Example:** Table 15.3 demonstrates an example of a valid solution for the four-node channel allocation problem shown in Figure 15.5. The initial population has been created by using different permutations of the nodes in the chromosomes, e.g., 1, 2, 3, 4, or 1, 3, 2, 4, or 3, 4, 1, 2, or 3, 1, 2, 4. The fitness function is decided based on the total number of channels allocated, that is, the value of \( M \). The objective is to minimize this value of \( M \). For chromosomes with equal \( M \) values, the one with more 0s is selected as the better one. The reason is that the chromosomes having more 0s allow more channels to be added, while satisfying the interference constraint. Now, out of the \( P \times N \) columns, \( P \times N \times \rho \) columns are selected at random, where \( \rho \) is the probability of mutation. From a selected column, randomly a 0 and 1 is chosen and flipped. An unavailability of a column with 0 or 1 leads to a mutation failure. If the mutation results in a row with all 0 values, a leading 9 is placed, reducing the bandwidth requirement by 1. Simulation results show that the lower bound for the minimum number of frequencies range from \( \sim 300 \) to 500 for different types of demands. The GA approach takes only 1 to 3 generations (fraction of a second) to converge in a near-optimal solution.

**Table 15.3 Examples of Valid Solutions**

<table>
<thead>
<tr>
<th>solution</th>
<th>+1</th>
<th>-1</th>
<th>+1</th>
<th>-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>+1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>+1</td>
<td>-1</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>+1</td>
<td>-1</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>+1</td>
<td>-1</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
</tr>
</tbody>
</table>

\[ C = \begin{bmatrix} 5 & 4 & 0 & 0 \\ 4 & 5 & 0 & 0 \\ 0 & 0 & 5 & 0 \\ 0 & 1 & 2 & 5 \end{bmatrix} \quad D = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 3 \end{bmatrix} \]

**Figure 15.5** A four-node channel allocation problem.
### 15.5 Call Admission Control

An important issue in cellular systems is to maintain a good service quality. Admission control policies are designed to meet this end. A wise admission decision for new calls can improve the overall service quality by guaranteeing QoS to the already admitted calls, while having a minimal average blocking rate for new calls. An admission policy is a binary decision controlling the admission criteria. Thus, any such policy can be represented by a finite number of logic functions forming the policy space. However, the enormously large policy state space often needs some compact coding [4] to restrict the blow up of search space. There is a finite possibility for the coded policies to get stuck at local optimal solutions. Multimodal optimization strategies are required to solve such problems. Genetic algorithms provide such a robust optimization strategy that yields quite satisfactory results even when little is known about the search space.

Following Reference 4, a network system \( S \) with \( n \)-dimensional state vector \( s = (s_1, \ldots, s_n) \) is considered. The entire set of customers is divided into different classes depending on their relative importance. A state represents the number of customers of each class in the system. It has been shown in Reference 4 that for a system described by a finite dimensional Markovian state vector, the call admission control can be formulated as a Markov Decision Process (MDP). Such a model assumes a set of state-dependent controls. Fixing a particular policy actually specifies the state transition matrix, a set of controls and rewards depending on both the system state and state transitions. At every iteration, the current policy is evaluated and using this evaluation a better policy is obtained. Each step however requires solving a set of \( n \) linear equations. Each possible decision for each possible state is also evaluated. For this admission control, the strategy needs \( O(n) \) evaluations per iteration. The computational effort becomes excessive when the number of states is very high. Hence, GA has been used to reduce the computational complexity and yield moderately good results. For this, a suitable coding is needed to represent such policy-based decisions suitable for using in GAs.

If \( \eta \) represents an admission-decision and \( \eta \Gamma \) denotes the number of bits necessary to represent a policy, then the total number of possible policies is \( 2^{\eta \Gamma} \). For smaller values, this policy can be directly represented by the string called direct coding. However, for any practical system, the enormous size of search space makes such direct coding impossible and intractable. A compact coding strategy of good strings, which is also theoretically capable of coding any arbitrary string, is presented in Reference 4. It introduces two different policy coding procedures. In block coding, a table of entries is built, with each entry corresponding to a \( K \)-bit pattern. A number of such \( K \)-bit-patterns can be concatenated to form full policy strings of length \( \Gamma \eta \). The policy is now specified by the sequence of table indices, rather than sequence of \( K \) bits. For any \( G < K \) a compression is achieved, and the total number of entries is \( 2^G \). In program coding the states of the policies are assumed as inputs to a computer program. The policy coding now specifies the bits of the program. Assuming infinite time and memory, the existence of a universal computer or a Finite State Machine (FSM) can be modeled, which will provide the decision as the output.

These coded strings are represented by the chromosomes. A policy search scheme for two different classes of policies are modeled using Markovian queuing systems [4]. If \( \lambda_1, \lambda_2, \varrho_1^k, \varrho_2^k \) represents the arrival rate and blocking probability of two different classes of calls, then for any policy \( k \), the average blocking is given by \( \Pr[\text{block}] = (\lambda_1 \varrho_1^k + \lambda_2 \varrho_2^k) / (\lambda_1 + \lambda_2) \). Minimizing this \( \Pr[\text{block}] \) is essentially maximizing \( (1 - \Pr[\text{block}]) \). This measure provides the performance of any policy \( i \). For each generation \( g \), the performance \( \text{perform}_i \) of a policy \( i \) in the population is evaluated and the minimum \( \text{perform}_i \) is computed. The fitness function of policy \( i \) used is given by \( \text{fitness}_i = \exp(-\alpha(\text{perform}_i - \text{perform}_{\text{min}})) \), where \( \alpha > 0 \) is a constant scale factor constant that emphasizes or de-emphasizes deviation from the best population performance, \( \text{perform}_{\text{min}} \). Experimental results show that it is useful to increase \( \alpha \) with the increase of \( g \). Normal crossover (swapping the portions of parent policies) and mutation (flipping the bits) operations are used to generate the offspring policies. The algorithm is capable of obtaining (near-) optimal solutions within \( g = 1000 \) generations. Table 15.4 provides some sample results of direct, block, and program codes for 10 and 50 cells respectively. The table demonstrates that the GA approach achieves a near-optimal value of the blocking probability in almost all cases. Obviously, the time, iteration and...
length of strings varies for different coding strategies. Program codes perform the best in terms of the number of generations and string lengths, followed by the block codes and direct codes.

An adaptive resource allocation and call admission control scheme, based on GA, was proposed in Reference 19 for wireless ATM networks. Multimedia calls have their own distinct QoS requirements (e.g., cell loss rate, delay, jitter, etc.) for each of their substreams. The network usually allocates an appropriate amount of resource that constitutes a certain QoS level, which remains fixed during a call. But such static schemes are inefficient in terms of resource utilization. In the adaptive algorithm proposed in Reference 19, each substream declares a range of acceptable QoS levels (e.g., high, medium, low) instead of just a single one. With the variation of network resources, the algorithm selects the best possible QoS level that each substream can obtain, while achieving maximum utilization of the resources and their fair distribution among the calls. For example, in case of congestion, the algorithm tries to free up some resources by degrading the QoS levels of some of the existing calls. The problem essentially boils down to finding the best QoS levels for all existing calls amidst a large search space. Thus, if three types of streams, namely audio, video, and data are considered with four possible QoS levels — “High,” “Medium,” “Low,” and “No” Component — then a total number of $4^3 = 64$ QoS levels are possible, and if we consider $N$ existing calls, the search space is given by $64^N$. For $N = 10$, the dimension of search space will be $64^{10} = 1.153 \times 10^{18}$.

A GA has been provided in Reference 19, which is a tool for searching an optimal solution in this huge search space. Simulation results show that the algorithm is capable of searching solutions with a huge gain (250%) in terms of the number of admitted calls, while achieving a resource utilization ranging between 85.87% and 99.985%.

### 15.6 QoS-based Multicast Routing Protocol

A majority of multimedia applications, such as video on demand and conferencing, depend on efficient multicast protocols. Multicasting is essentially a selective broadcast of information from a source to a given set of destinations. The information is delivered from the source to the destinations using a multicast routing tree computed by multicast routing algorithm. Since the basic unicast routing (route computation from a single source to a single destination) with multiple QoS constraints is an NP-complete problem [20], the computation of a multicast route that satisfies certain QoS constraints is also computationally intractable. In wireless networks, fluctuations in available resources complicates the problem further. Thus, multicast routing in wireless networks, with multiple QoS constraints and imprecise information on the available network resources need some kind of heuristic-based technique to compute an efficient multicast routing tree. In this section, we describe the design and performance analysis of a multicast routing QoS-based multicast routing scheme [21], that relies on the link-state advertisements and uses a GA framework for near-optimal multicast delivery tree computation.

The problem thus is to compute a source-specific tree-based route, spanning the source and a given set of destination nodes, such that the route satisfies multiple QoS parameters, such as minimum bandwidth and end-to-end delay, and also complies with policy requirements such as minimum total bandwidth utilization. Modeling the input network as a graph, with the routers as the nodes and the network links as the edges of the graph, maps the QoS-based multicast routing problem to a constrained Steiner tree computation problem, which is a well-known NP-hard problem [1]. In case of wireless networks, it is rather difficult to obtain accurate values for these QoS parameters at any instant of time due to the

### Table 15.4 Results of Admission Control

<table>
<thead>
<tr>
<th>Cells</th>
<th>Null policy</th>
<th>Best policy</th>
<th>Direct code (GA)</th>
<th>Block code (GA)</th>
<th>Program code (GA)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0.121</td>
<td>0.0753</td>
<td>0.076 400 132</td>
<td>0.077 200 100</td>
<td>0.076 24 &lt; 40</td>
</tr>
<tr>
<td>50</td>
<td>0.216</td>
<td>0.072</td>
<td>0.097 2652 3600</td>
<td>0.078 510 800</td>
<td>0.072 48 &lt; 40</td>
</tr>
</tbody>
</table>
time-varying nature of the wireless medium and the dynamism in the available resources. However, the moments of their probabilistic distributions can be obtained by studying the history of their change over a certain period of time, broadcasted with link state advertisements. Proper modeling with probability distributions for each QoS parameter enables us to estimate the probabilistic bound corresponding to a particular value of that parameter. This bound serves as the guarantee provided by the network, for satisfying a particular value of a QoS parameter. Additionally, the tree should admit the maximum number of sessions with such guarantees. This is achieved by efficient resource and traffic management mechanisms.

15.6.1 Underlying Routing Algorithm

The network is represented by a graph \( G = (V, E) \), where \( V \) is a set of nodes and \( E \) is a set of links between node pairs. A logical path between two nodes \( v_1 \) and \( v_n \) is represented by \( v_1, v_2, \ldots, v_n \), where \( v_i \in V \) for \( 1 \leq i \leq n \). There can be multiple such paths between each pair of nodes. The primary objective is to develop an efficient scheme that will find a near-optimal multicast routing tree with source as the root and the destination as the leaves, such that

- The bandwidth provisioning for the specified QoS requirement is guaranteed.
- The end-to-end delay requirement is satisfied with maximal guarantee.
- The amount of bandwidth utilized is minimal.

The GA-based solution is encoded as follows:

- All the possible paths between the source and each destination are first computed and stored in a pool. Note that a path is a sequence of network nodes. Among these paths, we select those paths that conform to the bandwidth requirement.
- From each created pool, one solution is randomly chosen and concatenated with each other to obtain a multicast solution.

In order to achieve optimality in terms of bandwidth availability, bandwidth utilization, and the end-to-end delay guarantees, we devise an optimization function, which the GA can use in its course of action. The transmission delay along each link is assumed to follow an exponential distribution. Hence, the available bandwidth has been assumed to follow Poisson distribution [22] and the end-to-end delay on a path from the source to any particular destination is assumed to follow Erlang-K distribution [22]. Based on these assumptions, the probability distribution function for the end-to-end delay along a path from the source to a single destination is given as \( D_{\text{path}}(t) = \left( \frac{\hat{d}^{n-1}e^{-\hat{d}t}}{(n-1)!} \right) \), where \( \hat{d} \) is the average delay on each link and \( \kappa \) is the number of links on the path. Similarly, the probability of getting a bandwidth of \( B \) over a link \( l \) is given by the probability density function, \( \hat{b}(B) = \left( \frac{\hat{b}B^{\hat{b}-1}e^{-\hat{b}B}}{(\hat{b}B)!} \right) \), where \( \hat{b} \) is the average bandwidth on each link. In addition to these measures, we have considered residual bandwidth as the third measure of optimization. This is given by \( \sum_{l \in T} (\varsigma_l - \sigma_l) \), where \( \varsigma_l \) is the capacity of a link \( l \in T \), and \( \sigma_l \) is the bandwidth allocated for all the paths in the multicast tree \( T \), along the link \( l \). The optimization function is chosen as the linear combination of the probabilistic measure of the three QoS parameters. Hence the fitness function is given by:

\[
F = \Pi_{\text{path} \in T} D_{\text{path}}(t) + \Pi_{l \in T} \hat{b}(B) + \frac{\sum_{l \in T} (\varsigma_l - \sigma_l)}{\sum_{l \in T} \varsigma_l}.
\]

Genetic algorithm-based operations, namely selection, crossover, and mutation operations are then successively applied to the members of the population until there is little change in the quality of the best solution in the entire population. The best solution replaces the worst solution of the previous population. In crossover operations, the corresponding parts of two randomly selected multicast routing trees are concatenated to obtain new routing trees. In mutation operations, a path in a multicast routing tree is randomly selected and is replaced by another valid path between the same source and destination.
Illustrative Example: To illustrate the coding scheme of potential solutions, we study a network of eight nodes, shown in Figure 15.6, where node 1 is the source of multicast delivery and 4, 5, 6, and 7 are the destination nodes. Then the pools of valid paths for each source–destination pair are as shown in Figure 15.7. The initial population, as required by the GA, is created as follows. Each member of the population is formed by randomly selecting a path between each source–destination pair and then concatenating them to represent a multicast tree spanning the source node and the set of destination nodes. Figure 15.8 depicts the multicast delivery tree computed by the underlying routing algorithm for the sample network, shown in Figure 15.8, with node 1 as the source and nodes 4, 5, 6, and 7 as the destination nodes.

15.6.2 Improvement

The GA framework described here, combines the three QoS objectives into a single linear fitness function. This scheme works well when only one solution is needed. But when multiple, mutually conflicting optimization parameters are involved, it cannot yield solutions, which are better than the other with respect to a
single optimization parameter, but not the superior when all the optimization parameters are considered. In such a case, no solution is dominated by others when all the parameters are considered. These are generally termed as nondominated or pareto-optimal solutions, generated by using a multi-objective GA. The GA-based multicast routing algorithm has been extended to incorporate a multi-objective QoS-optimization mechanism [8]. The procedure does not combine the three predefined QoS parameters into a single objective function but attempts to optimize each parameter individually, thereby providing a near-optimal and non-dominated set of solutions (i.e., multicast trees). The solution set consists of not only those trees that offer best delay, bandwidth requirement, and residual bandwidth guarantee individually, but also a set of trees compromising fairly between the three optimization parameters.

Simulation results demonstrate that the algorithm is capable of obtaining a near-optimal multicast tree in reasonable time. With a session arrival rate of 5–10 multicast sessions, the average session blocking rate is only 2–5%. The multi-objective GA improves the flexibility of this scheme by offering a set of nondominated solutions. The user now has the flexibility to choose his/her favorable solution from this nondominated set. The dynamism and fluctuation of networks always creates resource uncertainty. This might result in the unavailability of a particular QoS-constrained path. The nondominated solutions aids in offering an alternate QoS-guaranteed path, thereby resulting in the graceful degradation scheme of QoS provisioning. With multi-objective GA, it becomes possible to sustain more calls with their minimum level of QoS.

15.7 Conclusion

In this chapter, we have presented a survey of the computationally difficult problems specific to wireless networking that have been solved using GA, a bio-inspired optimization algorithm. The primary objective of each of the problems is to derive a near-optimal solution in a computationally feasible time. GA provided a tool for deriving such solutions. Relevant performance results of the solutions for the problems have been presented to illustrate the efficiency of GAs in solving the complex problems. Potentially, GA can be used in any of the optimization problems faced in the design and operation of wireless networks. While, in this chapter, we have predominantly discussed problems from the cellular network domain, recent developments in the area of ad hoc and sensor networks are posing new challenges, a majority of which are difficult optimization problems. As a result, there have been some research attempts [23–25]
to solve these problems using GA. We hope that this chapter will help in providing a clear idea on the methodology of solving complex, real-life problems in the wireless domain using GAs.

References


