Optimizing Parameters of a Mobile Ad Hoc Network Protocol with a Genetic Algorithm

David Montana and Jason Redi
BBN Technologies
10 Moulton Street, Cambridge, MA 02138
{dmontana,jredi}@bbn.com

Abstract

Mobile ad hoc networks are typically designed and evaluated in *generic* simulation environments. However the real conditions in which these networks are deployed can be quite different in terms of RF attentution, topology, and traffic load. Furthermore, specific situations often have a need for a network that is optimized along certain characteristics such as delay, energy or overhead. In response to the variety of conditions and requirements, ad hoc networking protocols are often designed with many modifiable parameters. However, there is currently no methodical way for choosing values for the parameters other than intuition and broad experience. In this paper we investigate the use of genetic algorithms for automated selection of parameters in an ad hoc networking system. We provide experimental results demonstrating that the genetic algorithm can optimize for different classes of operating conditions. We also compare our genetic algorithm optimization against hand-tuning in a complex, realistic scenario and show how the genetic algorithm provides *better* performance.

1 Introduction

There are many situations where a data network is required in places where there is no fixed networking infrastructure and no time to create such an infrastructure. Examples of such situations occur in military operations, law enforcement, and rescue operations. *Ad hoc networking*, the ability to form a network dynamically from scratch using wireless connections, addresses this need. When the nodes of the network are mobile, as is usually the case, the networks formed are called *mobile ad hoc networks*, or MANETs. The dynamic nature of MANETs provides special challenges beyond those in standard data networks [5, 9]. Not only must the network form initially, but it must maintain itself as the nodes move around and as transmission properties change. With changing connectivity between nodes, including the entrance and exit of nodes, the network must have the ability to adapt its topology to reflect these changes.

The networking protocol governs not only how the nodes communicate the data they need to send each other but also how to dynamically form and reform the network. Mobile ad hoc networks (MANETs) require special protocols; a review of a range of such protocols is given in [13]. One challenging aspect of protocol design is that the ideal behavior of a protocol differs based on the situation (i.e., the operating conditions and the goals). As an example, consider the case when the nodes are moving very quickly so that which pairs of nodes are within communications distance of each other is constantly changing. In this case, the network should generally incur the overhead of probing for neighbors and reconfiguring the network topology quite often. In contrast, consider the case when the nodes are relatively stationary and the communications capabilities between any two nodes is slow and rare to change. In this case, the network should be quite conservative and slow in sending out periodic neighbor discovery packet, and reacting to possible link changes.

In this paper, we focus on a particular protocol designed for use with nodes that have directional antennas [11, 10]. While it is a specialized protocol, it is similar in its basic operation to other more standard protocols, such as optimized link state [2]. Of particular interest for this work is the fact that it has a variety of parameters that can be adjusted to change its behavior. This provides the flexibility to adapt the protocol's behavior to different environments and situations. Below, we discuss further the algorithm and the parameters that can be adjusted (Section 2), a genetic algorithm to select values well suited to a particular situation (Section 3), and experiments that demonstrate its effectiveness (Section 4).

A large amount of work has been done on the application of genetic/evolutionary algorithms (as well as other stochastic search algorithms such as simulated annealing and ant colony optimization) to communications networks. We will not attempt to summarize all this work, but do refer the interested reader to [15] (which is complete but now a bit out-of-date) and [7] (which is more recent but less complete) for pointers into the literature. Some research specifically on applying genetic algorithms to ad hoc networks includes the following. [16] investigates using a genetic algorithm to create routing tables for an underwater ad hoc network. [12] and [1] examine the application of genetic algorithms to dynamically optimized routing in MANETs. [17] looks at how to use a genetic algorithm to cluster network nodes into subnetworks. However, our work is unique in providing the capability to adapt the behavior of a MANET to different operating conditions.

2 The Protocol and Parameters

The particular protocol we used in our work is complex and described in [11, 10]. While we will not explain the full protocol here, we provide a brief overview that includes enough detail to understand the operation of the particular parameters we tuned in this work.

The network protocol is of the family of "proactive link state protocols". This means that the protocol sends out periodic broadcast heartbeat packets in order to alert potential neighbors of a node's existance. If a node detects "enough" of another node's heartbeats, then we define that a communications "link" exists between the two nodes. If we stop detecting heartbeats from a neighboring node for a particular amount of time, we consider the link to be broken. Since the ability to successfully receive a broadcast packet is a complex probabilistic function of RF fading, pathloss, and network congestion, we perform an averaging over time before declaring an actual change in a link state. When a link goes up or down, we trigger the transmission of a link state update (LSU) packet, which is flooded throughout the network. The LSU packet describes all the (one-hop) links that a node thinks it has. By distributing these packets thoughout the network, each node can have a picture of the current topology and be able to determine the shortest paths between any two nodes in the network.

Since the processes of detecting the state of the link and distributing a changed link state take up bandwidth in the network, it is undesireable to do this unnecessarily. However, determining the best rates and thresholds for these protocols, given a network where the traffic, topology, and paths are regular changing, is a difficult task. It is common for someone to hand-tune a particular scenario, only to find that this particular tuning causes poor performance at another time or place in the network.

We now provide a list of those parameters that we allowed the genetic algorithm to adjust in order to configure the networking protocol for a particular set of operating conditions:

- **Heartbeat Interval** is how often to send neighbor discovery heartbeats (used for detecting new neighbors as well as a lost link to an existing neighbor).
- **Heartbeat Points** is the number of *points* to assign each received heartbeat from a neighbor node.
- **Score History Size** is the window of time to observe for making decisions about scores/points.
- **Up Score Threshold** is how high a score is needed within the history window to bring up a link to a new neighbor.
- **Down Score Threshold** is how low the score must go before the link to an existing neighbor is torn down.
- **Routing Algorithm** is either Hazy Sighted Link State [14] or standard link state.
- **Routing Event Interrupt Period** is how often the routing module checks for link changes. If a link has changed, a new routing table is computed and a changed link state update packet is flooded through the network.
- **Routing Global Interrupt Period** is how often to send a new link state update (LSU) regardless of whether the state has recently changed or not. This is done for refreshing previous LSUs and insuring that previous LSUs are not lost.
- **Traffic Max Attempts** is the number of (re)transmission attempts to use at the radio layer for user traffic.

For all the experiments described in this paper, we constrained the nodes to all use the same set of parameter values, so we need to select only a single set of parameter values for the network. [Note that we have done some experiments not described here where nodes of different varieties (helicopters and ground vehicles) could use different parameter values.]

As a simple example of how optimal parameter values vary based on the operating conditions, consider again the two scenarios introduced above. In the first scenario, the nodes are on average relatively close together but moving quickly relative to each other. In the second scenario, the nodes are relatively distant from each other but moving slowly. In the first case, the network needs to bring links up and down quickly, and hence should potentially utilize more frequent heartbeats, less averaging over time, and less stringent thresholds to pass for bringing links up and down. In the second case, the networking algorithm should potentially use more averaging over time to overcome noise, less frequent heartbeats for less overhead, and more stringent thresholds.

What makes selecting a good set of networking protocol parameters difficult is the interaction between parameters. This is not as simple as just increasing or decreasing the value of a single parameter. Changing a parameter value in the right direction can make performance worse unless some other parameters are also adjusted the right amounts. (In biological terms, this interdependence between parameters is called *epistasis.*)

Further complicating the process of parameter tuning is the large amount of time required for evaluating the performance of a single set of parameters. For example, in the experiments described below, it required approximately 10 minutes to run a simulation to evaluate a parameter set. It is not uncommon for some ad hoc network simulations to take hours or even days per run.

While the parameter values are very difficult to tune by hand, the process of parameter tuning is amenable to automated optimization. One can associate a numerical score with the performance of the networking algorithm. (Since there may be multiple performance criteria, it may require combining multiple scores into a single score.) Hence, this is a standard function optimization problem since the goal is to optimize the value of the performance metric over the space of possible parameter values. Since it is possible to run the simulations to evaluate the different parameter sets in an automated fashion, the optimization process can proceed without human intervention.

The resulting optimization problem has the following properties:

- a discrete search space (because the parameters can only take on a discrete set of values)
- a nonlinear and potentially discontinuous objective function
- a rugged landscape, i.e. many local optima far from the global optimum

This limits which optimization algorithms are sensible ones to apply. Algorithms to avoid include gradient-based methods, such as Levenberg-Marquardt, Newton-Raphson, quasi-Newton, and conjugate gradient, because they degrade badly when applied to a rugged landscape rather than a function that is smooth and has a single peak (or a small number of local peaks). Stochastic methods such as simulated annealing, tabu search, and genetic algorithms perform much better with rugged landscapes. Genetic algorithms, because of their population-based approach spreading "probes" throughout the search space, are particularly robust with respect to search space ruggedness. Due to limited time and resources, we could only investigate one optimization algorithm, so we chose to use a genetic algorithm because of the confidence that it would work due to its robustness. However, in the future it is worth doing a comparison with an algorithm such as simulated annealing, which cannot handle large amounts of ruggedness but performs better if the search space is only moderately rugged. [Note that one other big advantage of genetic algorithms is the ability to parallelize them on a large scale by spreading the evaluations across different machine. This could be particularly important when doing optimization runs much larger than those described below.]

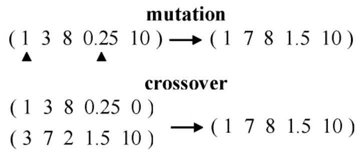

Figure 1: The genetic operators

3 The Optimization Algorithm

We now discuss some key aspects of the genetic algorithm we have defined to optimize the parameters of the networking protocol.

Representation - We use what is now the standard representation for parameter optimization problems, choosing the chromosome to be just a list of the parameter values. This is often referred to as a real-valued (as opposed to binary-valued) chromosome. For each parameter we are optimizing, we select a minimum value, maximum value, and step size. This defines a discrete set of possible values for the parameter.

Generating New Individuals - We initialize the population with purely random individuals. The genetic operators we use are the standard mutation and uniform crossover shown in Figure 1, with a probability of 0.5 for each. We use an exponential parent selection scheme and a steady-state replacement policy, adding a new individual into the population immediately after its generation and discarding the worst population member. A steady-state replacement strategy generally finds a solution much faster than a generational one because it can immediately exploit good individuals, and this is important given how long it requires to evaluate an individual.

Evaluation of an Individual - To evaluate an individual, our algorithm executes a simulation with the networking parameters set as given in the individual. During the simulation, it gathers statistics about networking performance, and afterwards it combines these into a single score.

The simulation uses OPNET Modeler with the OPNET Wireless Module, commercial software packages available from OPNET Technologies [8]. This software provides a high-fidelity model of a wireless network that allows the user to specify properties of the model including: RF propagation, interference, transmitter/receiver characteristics, and node mobility.

To complement the OPNET software, we created a scenario generator that accepts as inputs high-level specifications for node motion and network traffic. It creates point-by-point trajectories and packet transmission histories for each node that are randomly selected according to the specifications. The motion specifications include the dimensions of the area in which the nodes are constrained to move, the constant speed at which the nodes move, and the time spent stationary at waypoints. From these specifications, the scenario generator creates for each node a sequence of piecewise-constant-velocity legs by selecting random points in the area and connecting them, with stopover time at each point where the path changes direction. The scenario generator also creates a set of packets, each with a source node, destination node, and time, from some statistical network traffic specifications. Throughout a genetic algorithm run, we use a single scenario generated prior to the run. This scenario plus predefined RF propagation profiles and interference models provide a set of inputs to the simulator that are the same for each execution. Hence, differences in the performance are due solely to different values for the parameters.

For calculating the score, there are a variety of potential performance metrics. Some are based on quality of service (QoS), such as dropped packets and transmission delay. Others are not directly tied to QoS, such as power consumption rate. Hence, our problem is an example of a multiobjective optimization problem. There are a variety of different techniques for using genetic algorithms for multiobjective optimization [3]. We use the simplest and most common, create a single objective function as the weighted sum of multiple objective criteria. Selection of the values of the weights provides an explicit tradeoff between different criteria. (Varying weights across different optimization runs allows generation of different points on the Pareto optimal surface [6].) For the current work, we used a weighted sum of just two criteria: dropped packets and transmission delay. In fact, we ended up putting almost all the weight on the dropped packets score.

4 Experiments

We have performed a set of preliminary experiments that illustrate some of the properties of the optimization algorithm described above. In this section, we examine the data used for these experiments and the experimental procedure employed, and then provide some results and analysis.

4.1 Datasets

We used one particular dataset to compare automated optimization to manual tuning of parameters. This dataset is based on a live mobile network demonstration performed for the DARPA/Army Future Combat System Communication program at Lakehurst, NJ. In this demonstration, mobile nodes (SUVs) drove around a wooded area communicating with each other using ad hoc networking. A helicopter was also a node in the network, though it due to foliage and buildings it had communications with only about half of the ground nodes at any one time. As part of this program, we had carefully constructed a simulation scenario that modeled the terrian and network of this demonstration. Of particular interest is a 60-second snippet of this scenario which it proves particularly difficult for the networking algorithm to maintain good performance. A 60-second "introduction" with no data traffic has been added to this snippet to allow calibration by the networking algorithm. This enhanced snippet has been the focus of extensive efforts to hand-tune the parameters. Hence, using this dataset provides a good means to measure how well the automated approach compares with the manual approach.

We used a scenario generator to create additional datasets. While these artificially generated datasets are less realistic than the Lakehurst data, they provide a means to run controlled experiments investigating the performance of the optimization algorithm. The scenario generator produces randomized scenar-

ios with nodes moving and communicating according to certain specified criteria. One specifies the following as inputs:
• the number of nodes
• the size of the area over which the nodes move
• the propagation loss model
• the average data rates between nodes
• the speed with which the nodes move
and the scenario generator will create a scenario satisfying these specifications. Each generated node trajectory consists of a set of randomly selected waypoints in the given area with straight-line, constant-speed motion between the waypoints.

We used two different sets of specifications for the scenario generator. Both specifications have 20 nodes, use a log-distance propagation loss model [4] with exponent of 4, have network traffic which follow Poisson arrivial process with uniform source/destination distribution, and a square area of size 1200 meters by 1200 meters for the node to move according to the random waypoint model. The sole difference in the specifications is the speed of the nodes, one having nodes traveling very rapidly at 10 meters per second, and the other with nodes traveling slowly at 0.5 m/s. We have therefore called the specifications "speedy" and "poky". For each of these two specifications, we have generated two 100-second datasets. One of these is to use for training, i.e. for evaluation during optimization runs, and the other for testing the performance of the optimized parameter sets. We call the training sets poky1 and speedy1 and the test sets poky2 and speedy2.

4.2 Experimental Procedure

A big issue for executing experiments is the wall time it takes for an individual simulation in OPNET to complete. For example, one simulation takes on average approximately 7-8 minutes for Lakehurst data and slightly less for the scenarios generated by the scenario generator. Since an optimization run requires evaluation of 600-1000 different parameter sets, and hence the same number of executions of the simulation, an optimization run takes multiple days on a single machine. This limited the number of runs we could do, forcing us to settle for a single run of the genetic algorithm per experiment when we would otherwise have prefered multiple runs in order to perform averaging of performance statistics. (Recall that the genetic algorithm is a stochastic algorithm and gets slightly different results each time.)

We used the following values of the genetic algorithm parameters. The population size was 300. The weights in the scoring function defining the tradeoff between the fraction of dropped packets and the average delay in seconds were chosen so that the former was weighted ten times as heavily as the latter. Since the fraction of dropped packets is generally ten times bigger than the average delay (with the former usually betweeen 1 and 10 and the latter usually between 0.1 and 1), giving dropped packets ten times more weight makes this by far the primary metric of performance.

For the experiments on the Lakehurst data comparing automated to manual tuning, we were evaluating the effectiveness of training (i.e., selecting parameters for optimal performance on a known dataset) and so we just considered how well the parameters performed on the training data. However, for the experiments involving the datasets created by the scenario generator, the goal was to determine how well the parameters tuned to one (or more) scenario(s) perform in a different scenario. Therefore, we tested the performance of the parameters on test datasets different from those used for training.

4.3 Results and Analysis

As discussed above, we have used the Lakehurst data to compare manual parameter tuning to automated tuning, since we had previously spent much effort tuning to this dataset manually. The results are the following:

Approach	Dropped Packets	Average Delay
Manual	0.089	0.0089
Automated	0.042	0.0036

Clearly, automated parameter tuning produces better results. In fact, if we look at the progress of the automated run on the Lakehurst data shown in Figure 3, we can can see that we do not require any sort of directed optimization to beat manual tuning. With just the 300 completely randomly generated individuals of the initial population, the automated search already outperforms manual tuning. However, we can see in Figure 3, not only for this optimization run but also the runs described below, that the genetic algorithm does outperform random search, improving the score significantly after random search has lost momentum.

The remainder of the experiments investigate the ability of automated parameter tuning to generalize to data on which it was not trained. (A very common problem in statistical estimation is overfitting to the training data due to insufficient quantity and/or variety of training data relative to the number of parameters to be estimated.) Generalization to new scenarios is critical to the success of the networking algorithm

trained on	tested on			
	speedy	poky	speedy2	poky2
speedy	3.9%	N/A	3.9%	10.3%
poky	N/A	0.3%	16.8%	2.0%
speedy+poky	3.9%	0.9%	4.8%	0.6%

Figure 2: The performance (dropped packets) of parameter sets trained on different data

in the real world. An actual scenario is different from even the best model due to reasons ranging from an inability to precisely model propagation loss and network traffic to an inability to know precisely the trajectories of the nodes.

We derived three different sets of parameters, one trained on (i.e., optimized to) the poky1 dataset, one trained on the speedy1 dataset, and one trained using both poky1 and speedy1 (looking at the aggregate performance). Then, we tested all three of these parameter sets on poky2 and speedy2. The results are shown in Figure 2.

One conclusion that we can draw based on these preliminary results is that parameters trained on one scenario can generalize to statistically similar scenarios but not in general to statistically different scenarios. We observe that the parameters derived by training on speedy1 perform similarly well on speedy2 but do poorly on poky 2. Similarly, the parameters trained on both speedy1 and poky1 perform well on both test sets, speedy2 and poky2. The parameters trained on poky1 do perform poorly on speedy2 as expected, but in an apparent anomaly also have relatively poor performance on poky2. This anomaly is explained by the fact that the nodes are moving so slowly that they remain in essentially the same geometric configuration for the entire duration of the scenario. So, the parameters are specifically tuned to the geometric configuration of the poky1 scenario, and cannot generalize to the new geometry of the poky2 scenario. (In the speedy1 scenario, the nodes are covering a wide variety of different geometric configurations, providing a more diverse training set, and hence leading to better generalization.)

A second conclusion that we can draw from these results is that more variety in the training set leads to parameters more robust over a range of different opearting conditions (but potentially less well tuned to specific conditions). This robustness is illustrated by the parameters trained on the aggregate of the two training sets (speedy1 and poky1) doing well on both test sets (speedy2 and poky2), and much better on poky2 than even the parameters trained on poky1. This stands in contrast to the poor performance of parameters trained on one type of scenario when tested on the other type. The experimental results do not show that the parameters trained over a wider range of opearting conditions perform worse in particular situations than those trained specifically for those conditions. However, based on past experience of ours (e.g., [6]) and others optimizing parameter values, we do have reasonable expectations that such a performance degradation will occur if we try to make one set of parameters cover too many different types of operating conditions.

5 Conclusions and Future Work

The first conclusion we can draw from our work is that the values chosen for the different parameters of the networking algorithm make a big difference in the performance of the network. Furthermore, there is not a single set of parameters that is the best, since the performance of a set of parameters depends greatly on the conditions under which the network is operating. This makes the problem of selecting a good set of parameters an important and difficult one.

A second conclusion is that automated parameter optimization produces significantly better parameter values than hand tuning, at least based on our preliminary experiments. Hence, the automated approach is one well worth pursuing to greater levels of sophistication.

A third conclusion is that for automated parameter optimization to work best, the training data should represent the full range of operating conditions under which the parameters need to function. This need for sufficient quantity of representative training data is one that is not unique to this problem but is rather common to all forms of statistical estimation.

The work we have described is just preliminary and suggests some possible future work. One potential future task is comparing the genetic algorithm optimization performance with other stochastic optimization algorithms such as simulated annealing or tabu search. While we only had opportunity to investigate one algorithm so far, it would be valuable to do a comparative study of optimization techniques.

A second future task is implementation of the capability to have multiple machines running evaluations of different parameters simultaneously. An important feature of genetic algorithms is the ability to achieve approximately linear speedups via parallel evaluations (e.g., [6]), and we should exploit this to speed the optimization runs.

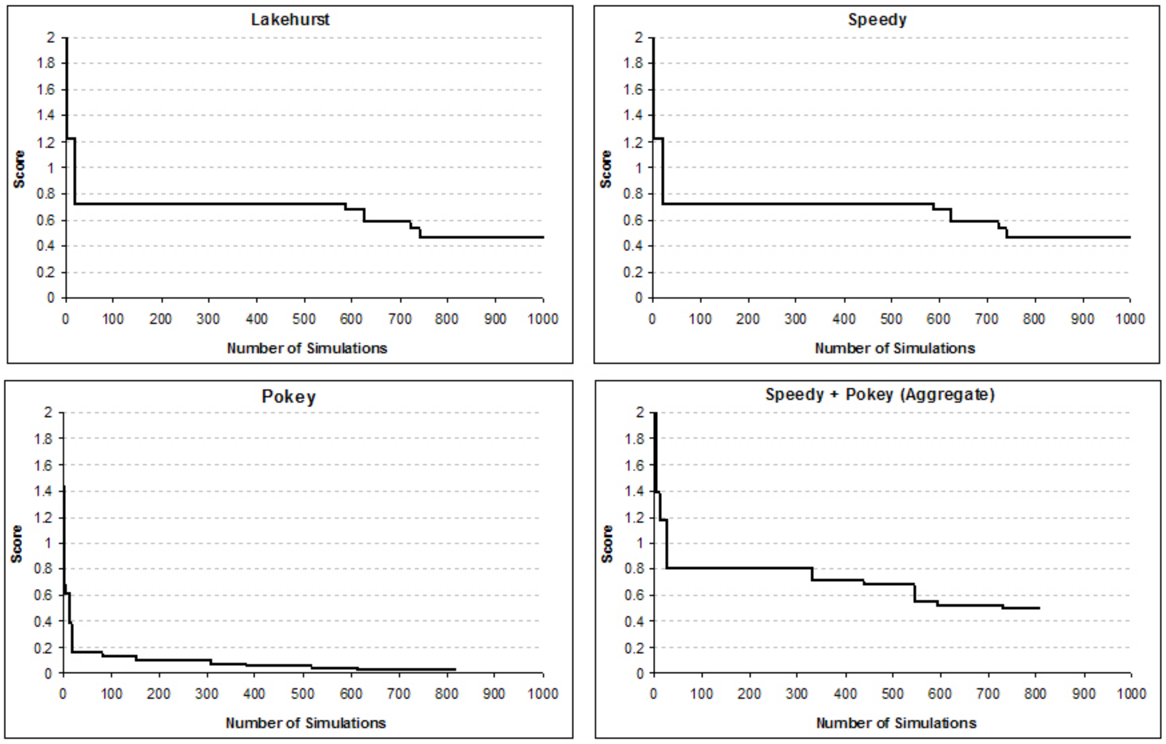

Figure 3: Progress of automated optimization on four different datasets

Another future task is trying the parameter optimization, which is performed using simulation, in real networks. The need to try out hundreds of different parameter value combinations in a scenario where everything else remains the same means that simulation is a necessary piece of the optimization process. However, we should do experiments with how well parameters optimized in simulation perform in a real network. Furthermore, we should investigate how best to measure the conditions of a real network so that we can capture them in simulation. Potentially we could even have the parameters of a real network optimized in real time.

6 Acknowledgements

This work was sponsored by DARPA/IPTO under contract number DASG60-02-C-0060. Content of this work does not necessarily reflect the position or the policy of the Government, and no official endorsement should be inferred.

References

[1] C. Ahn and R. Ramakrishna. A genetic algorithm for shortest path routing problem and the sizing of populations. *IEEE Transactions on Evolutionary Computation*, 6(6):566–579, 2002.

[2] T. Clausen and P. Jacquet. Optimized link state routing protocol, rfc 3626 (experimental), 2003. http://www.ietf.org/rfc/rfc3626.txt.

[3] C. Coello. A comprehensive survey of evolutionary-based multiobjective optimization techniques. *Knowledge and Information Systems*, 1(3):269–308, 1999.

[4] J. D. Gibson. *The Communications Handbook*. CRC Press, 1997.

[5] Z. Haas, J. Deng, B. Liang, P. Papadimitratos, and S. Sajama. Wireless ad hoc networks. In J. Proakis, editor, *Wiley Encyclopedia of Telecommunications*. John Wiley and Sons, 2002.

[6] D. Montana. Automated parameter tuning for interpretation of synthetic images. In L. Davis, editor, *Handbook of Genetic Algorithms*, pages 282–311. Van Nostrand Reinhold, 1991.

[7] D. Montana and T. Hussain. Adaptive redesign of reconfigurable data networks using genetic algorithms. *Applied Soft Computing*, 4(4):433–444, 2004.

[8] Opnet Technologies, Inc. Wireless module, 2004. http://www.opnet.com/products/modules/wireless_module.html.

[9] R. Ramanathan and J. Redi. A brief overview of ad hoc networks: challenges and directions. *IEEE Communications Magazine*, 40(5):20–22, 2002.

[10] R. Ramanathan, J. Redi, C. Santivanez, D. Wiggins, and S. Polit. Ad hoc networking with directional antennas: A complete system solution. *Proceedings for the 2004 IEEE Wireless Networking and Communications Conference (WCNC 2004)*, 2004.

[11] R. Ramanathan, J. Redi, C. Santivanez, D. Wiggins, and S. Polit. Ad hoc networking with directional antennas: A complete system solution. *IEEE Journal on Selected Areas In Communications*, to appear, March 2005.

[12] A. Roy and S. Das. QM2RP: A QoS-based mobile multicast routing protocol using multi-objective genetic algorithm. *Wireless Networks*, 10(3):271–286, 2004.

[13] E. Royer and C.-K. Toh. A review of current routing protocols for ad hoc mobile wireless networks. *IEEE Personal Communications*, 6(4):46–55, 1999.

[14] C. Santivanez and R. Ramanathan. Hazy sighted link state (hsls) routing: A scalable link state algorithm. BBN Technical Memo BBNTM-1301, BBN Technologies, 2001.

[15] M. Sinclair. Evolutionary telecommunications: A summary. In *GECCO'99 Workshop on Evolutionary Telecommunications: Past, Present and Future*, pages 209–212, 1999.

[16] E. Sozer, M. Stojanovic, and J. Proakis. Initialization and routing optimization for ad-hoc underwater acoustic networks. In *Proceedings of Opnetwork*, 2000.

[17] D. Turgut, S. Das, R. Elmasri, and B. Turgut. Optimizing clustering algorithm in mobile ad hoc networks using genetic algorithmic approach. *Proceedings of IEEE Global Telecommunications Conference (GLOBECOM)*, 21(1):62–66, 2002.