Knowledge Sharing Agents Using Genetic Algorithms in Mobile Ad Hoc Networks

by

ELKIN URREA

A Dissertation Submitted to the Graduate Faculty in Engineering
in Partial Fulfillment of the Requirements for the
Degree of Doctor of Philosophy
The City University of New York

2010

This manuscript has been read and accepted for the Graduate Faculty in Engineering in satisfaction of the dissertation requirement for the degree of Doctor of Philosophy.

Date	Prof. M. Ümit Uyar Chair of Examining Committee
Date	Prof. Mumtaz Kassir Executive Officer

Prof. M. Ümit Uyar (Advisor)

Prof. Michael Conner

Prof. Jizhong Xiao

Dr. Ibrahim Hökelek

Mr. Giorgio Bertoli

Supervisory Committee

THE CITY UNIVERSITY OF NEW YORK

Abstract

Knowledge Sharing Agents Using Genetic Algorithms in Mobile Ad Hoc Networks

by

Elkin Urrea

Advisor: Prof. M. Ümit Uyar

We present a novel approach for knowledge sharing mobile agents to achieve a uniform spread of autonomous mobile nodes over an unknown geographical terrain. In this research, knowledge sharing agents adjust their speed and directions within a mobile ad hoc network (MANET) using a decentralized topology control mechanism based on genetic algorithms (GAs). The genetic information that each mobile agent exchanges with other neighboring agents within its communication range includes the node's location, speed, and movement direction.

We show that our GA-based topology control system can be modeled as a dynamical system in order to provide formalism to study its convergence trajectory in the space of possible populations. We apply this discrete time dynamical system model for calculating the cumulative effects of genetic operators such as selection, mutation, and crossover as a population of possible solutions evolves through generations using our GA-based approach.

To demonstrate applicability of our topology control algorithms to real-life problems and evaluate their effectiveness, we implemented a simulation software system and several different testbed platforms. The simulation and testbed experiment results indicate that, for important performance metrics such as normalized area coverage (NAC) and convergence rate, our algorithms can be effective in deploying nodes under restrained communication conditions in MANETs operating without prior geographical terrain knowledge. Since our topology control algorithms adapt to any local environment rapidly and do not require global network knowledge, they can be used as real-time topology controllers for realistic military and civilian applications.

Acknowledgments

This research project would not have been possible without the support of many people. The author wishes to express his gratitude to his Ph.D. supervisor, Prof. Dr. M. Ümit Uyar who was abundantly helpful and offered invaluable assistance, support and guidance. The author would like to express his special thanks to the members of the supervisory committee, Prof. Michael Conner, Prof. Jizhong Xiao, Dr. Ibrahim Hökelek and Mr. Giorgio Bertoli for their guidance.

Deepest gratitude are due to all his friends, especially group members Mr. Cem Şafak Şahin, Mr. Janusz Kusyk, Mr. Stephen Gundry and Mr. Cevher Doğan for their friendship and invaluable assistance.

The author would also like to convey thanks to Prof. Uyar for providing the financial means through his grants from the U.S. Army Communications-Electronics RD&E Center grant W15P17-09-C-S021, National Science Foundation grants ECS-0421159 and CNS-0619577, and to the City College of New York for offering the laboratory facilities.

The author wishes to express his love and gratitude to his beloved family, especially his wife, Mariana Rojas, for their understanding and endless love throughout his studies.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Autonomous systems represent a blend of software and machinery to create intelligent platforms for complex real world problems without human control and guidance. These systems must be self-sufficient and capable of adapting their behavior to rapidly changing and most likely unfamiliar environments.

A mobile ad hoc network (MANET) consists of an autonomous system of mobile nodes which dynamically form a network without any pre-existing structure. These mobile entities are geographically dispersed and equipped with wireless transmitters and receivers to communicate with each other within the MANET. Communication among the mobile nodes is generally established through a multi-hop routing due to the limited range of transmission capabilities of each individual node. Since the mobile nodes move arbitrarily in a MANET, the network topology may change dynamically and unpredictably. One way of maintaining a uniform distribution of mobile nodes

over any terrain is to provide the nodes with the ability to adapt their speeds and movement directions based on their local neighbor nodes and surroundings (e.g., the number of neighbors, neighbors locations, obstacles within node sensing range, etc). The motion ability is the main characteristic for the nodes in a MANET that facilitates numerous useful network features, including (but not limited to) the ability to self-deploy. Self-deployment is one of the key features in our approach since it provides capability to start from a compact initial configuration, then, the nodes in a MANET can propagate by spreading out through time. It also provides the inextricable relation to the physical world, the appropriate deployment of mobile nodes is very important for a successful completion of cooperative tasks.

We propose a bio-inspired topology control approach using genetic algorithms (GAs) to produce a uniform deployment of mobile agents, discover network layouts that maintain the communication connectivity among mobile nodes, and converge toward a uniform distribution over a two-dimensional area. Note that we use the terms `node`, `mobile node`, and `mobile agent` interchangeably to refer to the physical mobile entities in a MANET, each of which is running the bio-inspired topology control algorithms. GAs belong to a particular set of evolutionary algorithms that use techniques motivated by evolutionary biology such as inheritance, mutation, selection, and crossover [3]. GAs have become an important tool in machine learning, multiprocessor scheduling, and function optimization [4]. Although the mechanisms by which a GA functions have not been fully validated, GAs have been successfully employed at solving problems with no known deterministic algorithm alternative.

For example, one of the realistic solutions in robotics research is to use distributed intelligence such as in *swarm robotics* [5, 6], where large numbers of simple robots in multi-robot systems coordinate to obtain a collaborative behavior in order to complete a difficult task. GAs is one of the most studied approaches in order to find solutions for these types of collaborative and difficult tasks. GAs look for the best individuals, i.e., the best solution or optimum result in an entire problem set.

Our GA-based topology control algorithm, called FGA, uses a GA in the decision-making process of adaptive and autonomic systems to dynamically evolve reconfiguration strategies that balance competing objectives at run time. FGA adapts a mobile node's speed and direction according to its current environment (e.g., the number of neighbors and the movement profiles).

One can envision many applications for our GA-based topology control algorithms ranging from military to commercial applications, such as search and rescue missions (e.g., locating humans trapped in rubble after an earthquake), controlling unmanned vehicles and transportation systems, clearing mine-fields, and spreading military assets (e.g., robots, mini-submarines, etc.) under harsh and bandwidth-limited conditions. In these applications a large number of mobile nodes can gather information from multiple viewpoints simultaneously, allowing them to share information and adapt to the environment quickly. A common objective among these applications is the uniform distribution of autonomous mobile nodes operating on geographical areas without prior geographical terrain knowledge. The topology control of autonomous mobile nodes face extra challenges in MANETs since: (a) due to mobility, local terrain

may change dramatically in a short time-span during an operation, (b) the number of mobile nodes may increase or decrease unpredictably due to malfunctions, (c) mobile nodes may not have access to neither navigation maps or GPS devices, but can only have limited information collected from local neighbors, and (d) nodes may be deployed into a terrain from a single entry point (rather than random or other types of initial distributions often seen in existing research).

Markov chains offer an appropriate model to analyze our GA-based topology control algorithm [7, 8]. We build an ergodic homogeneous Markov chain to model the convergence properties of FGA extrapolated from a theoretical standpoint. The Markov chain demonstrates that FGA will both converge by analyzing Dobrushin's contraction coefficient and that that convergence will most likely be an optimal solution by analyzing the total fitness of the final stationary distribution. It analyzes these two metrics among varied communication ranges and geographic areas.

We also implemented a Java-based simulation software to evaluate effectiveness of FGA and its applicability to real-life problems. Using this simulation software, we study the performance of FGA with respect to different parameters such as area coverage, average fitness values, average traveled time, and convergence In addition, in Bio-inspired Computing Laboratory at the City College of New York, we built a testbed to study the convergence properties of various GA-based topology control mechanisms in MANETs.

The rest of this thesis is organized as follows. The related research in the area of

genetic algorithms, mobility models and deployment techniques are summarized in Chapter 2. A detailed description of our topology control mechanism is presented in Chapter 4. A dynamical system model to predict the behavior of FGA and analyze the effect of the initial population on the convergence of our algorithm is described in Chapter 5. FGA is modeled as a Markov chain using the mobile nodes' experimental behavior to demonstrate the underlying stochastic and convergent nature of the algorithm in Chapter 6. Our simulation experiments and the testbed implementations are given in Chapters 7 and 8, respectively. Chapter 9 concludes the thesis work.

Chapter 2

Related Work

Many distributed robotics applications utilize GAs for various decision making and optimization tasks. One approach utilizes a knowledge-based GA to find paths for mobile robots to move in complex and dynamic environments [9]. In another approach, autonomous mobile agents emulate the social behavior of insects collecting information from their environment using different sensors and a GA-based applications for movement [10]. Ref. [11] suggests a GA to optimize the speed and the distance among autonomous robots that simulate cars moving in a highway while avoiding collisions. Similarly, Refs. [12] and [13] use a genetic approach to accomplish distance-safety criteria for mobile robots in motion while avoiding obstacles. Ref. [14] compares a linear genome genetic programming system with the performance of hierarchical genetic programming methods by using simulated miniature robots to accomplish wall following, light avoidance, and obstacle avoidance. In a robotic urban search and rescue application [15], GA is used for searching a solution set for kinematic problem

space, such as using the tacks or wriggling like a snake to move forward. In a similar application [16], distributed autonomous robots aim to deploy in a terrain composed of multiple rooms. These robots run a GA in order to generate a solution set to decide the next movement for distributed autonomous robot teams, unlike our GA applications, they do not aim to distribute the robots uniformly. Ref. [17] considers mobile robots equipped with ultrasonic sensors in order to collect range-limited data from the environment where they spread. By contrast to our topology control algorithm, this approach finds an optimal position using a global map knowledge.

MANETs are characterized by many small processors that work together to perform complex tasks with no or little outside help. These systems need to be rapidly deployable and utilize protocols that allow the processors to automatically set up multihop communication paths among themselves. If the nodes are mobile, the network topology needs to change when the nodes move out of wireless range and may have to re-establish broken paths. Researchers are also considering swarm robotics to address this problem. Ref. [18] presents a multi-robot system that collectively try to find a path to all potential places to build a structure according to some set of user specifications. Similarly, a particle swarm optimization algorithm for path optimization of soccer playing robots is proposed in Ref. [19]. In Ref. [20], swarm optimization is used for route planning of UAVs using a fitness function based on both flight time and safety. Differently, Ref. [21] presents swarm micro-robots network to coordinate and realize different underwater tasks. A distributed navigation and exploration strategy is studied in Ref. [22], where the robots explore an environment with their sensors,

then memorize the data and place them together on a map.

In addition to swarm robotics, deployment of mobile agents has been considered by a number of authors in a variety of contexts. Ref. [23] uses a potential-field approach from mobile robotics to deploy mobile sensors. The fields are constructed such that each sensor is repelled by obstacles and other sensors, thereby forcing the network to spread itself throughout the environment. Ref. [24] considers autonomous dispersion of mobile nodes using a random diffusion method. In Ref. [25], mobile wireless sensors organize themselves in an adaptive manner over a geographical area by utilizing a self deployment algorithm based on cooperative mobile robotics. The percentage of the region covered, uniformity of network topology, time elapsed until all the nodes reach their final locations, and the average distance traveled by each node are considered as performance metrics. Ref. [26] uses Delaunay triangulation techniques to maximize area coverage while minimizing gaps and coverage overlaps by adjusting the deployment layout of nodes close to equilateral triangulation.

Mobility models represent the movement of mobile nodes, and how their location, velocity and acceleration change over time. Such models are frequently used for simulation purposes when new communication or navigation techniques are investigated. Different mobility models (e.g., random waypoint, random way group mobility, Manhattan mobility, and freeway mobility) have been studied for the movement of mobile nodes [27]. Ref. [28] proposes a technique, based on renewal theory, for analyzing mobility models in ad hoc networks and apply their method to random waypoint mobility. Ref. [29] provides relation between various mobility models and their ef-

fects on performance of ad hoc network. Refs. [1] and [30] implement a discrete-time random walk model to analyze the node link stability in MANETs. A node only has the capability of moving to a neighboring cell within one time unit. A discrete-time walk model formulates how a wireless link changes its state. The mobility model used in this thesis is first introduced by Hökelek et al. where a mobile node can move into one of the six neighboring cells, but its movement direction and speed are determined by a GA rather than a random walk [1, 31].

Applications of GAs on MANETs particularly concentrate on partitioning a MANET into multiple clusters. Ref. [32] utilizes a GA to divide a network into clusters which satisfy multiple objectives for the upper-layer protocols. This approach considers only temporal and spatial node connectivity, disregarding mobile node movement characteristics (e.g., direction, velocity and acceleration). Ref. [33] applies GAs to provide the optimum clustering of network nodes into subnetworks. Specifically, they apply GAs to their weighted clustering algorithm and demonstrate performance improvements such as less number of clusterheads and more evenly balanced load distribution on them. Ref. [34] suggests a new clustering strategy to execute energy conservation and topology management based on GAs. Ref. [35] presents a GA for QoS route discovery method in MANETs which helps locating a *good* set of routes quickly and reacting to the network dynamics promptly but sacrificing the optimality of the routes. Similarly, GAs are applied to automated routing parameter selection [36] and optimization of a unicast routing protocol [37] used in MANETs. Ref. [38] applies a parallel computing GA to allocate target points to multiple mobile nodes so that the

exploration time of the region of interest is minimized.

Schema theory , Markov chains, and dynamic system models are widely used to model and analyze the efficiency of GAs. Schema theory [3, 4] describes how *schemata*, templates specifying groups (sets) of similar chromosomes, are expected to propagate from one generation to the next. Ref. [4] formulates a schema theorem that gives a lower bound on the expected fraction of a population in a schema after one generation of a simple GA. Finite, discrete-time Markov chain models of GAs have been used successfully in the past to understand the complex dynamics of a simple GA. For example, Ref. [39] models the simple GA as a Markov chain, where the Markov chain states are populations. Ref. [40] extends this work to a *Random Heuristic Search* model of GAs where each individual of the next generation is selected from a probability distribution over individuals in the search space. Ref. [41] analyzes a GA using an elitist strategy on a Markov chain by assuming that the genetic operation in the generation changes is restricted to selection, crossover, and mutation, and by evaluating the eigenvalues of the transition matrix of the Markov chain.

GAs applied to large scale problems should avoid convergence towards an unwanted solution or a local optima. Ref. [42] investigates the convergence properties of GAs applied to fitness functions perturbed by multiple sources of additive noise. In Ref. [43], the eigenvalues of the transition state matrix from a Markov analysis are used to estimate the convergence rate. In Ref. [44], GA convergence time behavior is modeled by a Markov chain to show the effects of binary and higher cardinality representation of a search space.

Our GA-based topology control algorithm, called FGA, for self-spreading of mobile nodes in MANETs differs from the above cited approaches in several aspects. FGA gather very limited information obtained from the neighboring nodes and does not require manual tuning of key parameters or prior knowledge of the operative scenario. FGA has the basic self-* properties of autonomic computing, such as self-healing, self-configuration, and self-adaptation, giving rise to a fully decentralized algorithm. FGA uses an objective function inspired by the force-based distribution in physics where each molecule attempts to remain in balanced position and to spend minimum energy to protect its own position [45, 46]. Another significant difference is that we assume no prior knowledge of the geographical area or the obstacles. Finally, since FGA is fully distributed without a central controller or any other type of privileged entity, and each agent is using only the local information available, as shown in later sections, FGA is resilient to agent losses.

Chapter 3

Genetic Algorithms

Genetic algorithms (GAs) are numerical optimization algorithms inspired by both natural selection and natural genetics [3,4,47]. GAs use stochastic-based search techniques that can avoid local minima and that are very flexible due to their encoding and evaluation phases. The basic concept of GAs is to follow the principles of survival of the fittest by simulating the mechanisms of biological evolution such as reproduction, mutation, recombination, and selection. GAs have a tendency to succeed in an environment in which there is a very large set of candidate solutions and in which the search space is uneven and has many hills and valleys. The algorithms are simple to understand and the required software easy to write.

In most general form of GA a population is randomly created with a group of individuals (possible solutions) created randomly (Fig. 3.1). Commonly, the individuals are encoded into a binary string. The individuals in the population are then eval-

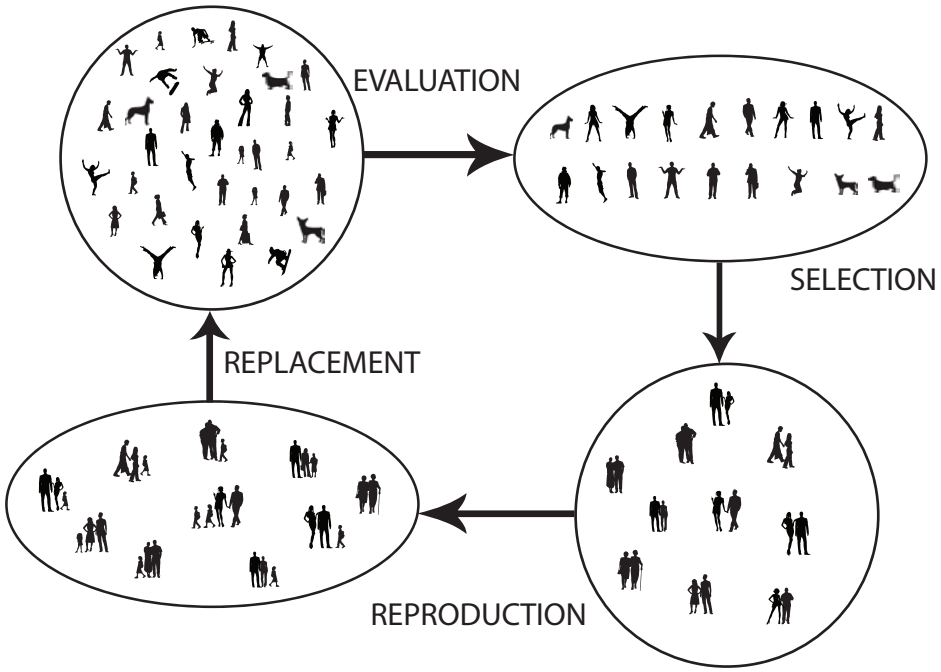


Figure 3.1: Basic form of Genetic Algorithm (GA).

uated. The evaluation function is given by the user which assigns the individuals a score based on how well they perform at the given task. Individuals are then selected based on their fitness scores, the higher the fitness then the higher the probability of being selected. These individuals then reproduce to create one or more offspring, after which the offspring are mutated randomly. A new population is generated by replacing some of the individuals of the old population by the new ones. With this process, the population evolves toward better regions of the search space. This continues until a suitable solution has been found or a certain number of generations have passed. A pseudo code for this simple GA is show in Algorithm 3.1.

The terminology used in GA is analogous to the one used by biologists. The connections are somewhat strained, but are still useful. The individuals can be considered to be a chromosome, and since only individuals with a single string are considered,

Algorithm 3.1 Pseudo-code of simple genetic algorithm.

```
t ← 0
initialize population P(t)
evaluate population P(t)
while not-termination condition do
  t ← t + 1
  select P'(t − 1) from P(t − 1)
  apply crossover on P'(t − 1)
  apply mutation on P'(t − 1)
  P(t) ← P'(t − 1)
end while
```

this chromosome is also the genotype. The organism, or phenotype, is the result produced by the expression of the genotype within the environment. In GAs this will be a particular set of unidentified parameters, or an individual candidate solution. These connections are summarized in Table 3.1.

Table 3.1: Comparison of biological and GA terminology.

Biological	GA
Chromosome or genotype	Structure, or string (often binary)
Locus	A particular (bit) position on the string
Phenotype	Parameter set or solution vector (real-valued)

3.1 Chromosome

A *chromosome* is the genetic material of an individual that represents the information about a possible solution to the given problem. In order to speed up a GA process in the mobile nodes, chromosomes should be selected carefully enough to include all the correct parameters and simple enough to reduce the computational processing power.

Binary strings are the most common representations of candidate solutions. In gen-

eral, a binary chromosome may be defined in a discrete search space Ω. A GA minimizes or maximizes a corresponding fitness function $f_i : \Omega \rightarrow \Re$. Then, each candidate solution in the search space Ω can be coded into l-bit binary chromosome that is an element of $\Theta = \{0, 1\}$. In other words, this is an one-to-one mapping $\Psi = \Omega \rightarrow \Theta$.

3.2 Genetic Operators

GAs mimic basic principles of life and apply genetic operators like selection, crossover (recombination), and mutation to a chromosome. The chromosome is constructed by a representation which assigns a string of symbols to every possible solution of the optimization problem. For selecting highly fit individuals for reproduction a large number of different selection schemes have been developed. The most common *selection* operator is *proportionate selection* (also called roulette wheel selection), where a chromosome is given a probability of being selected that is directly proportionate to its fitness. We can imagine the entire population forming a roulette wheel with the size of an individual's slot proportional to its fitness. The wheel is then rolled and the figurative ball thrown in. The probability of the ball coming to a rest in any particular slot is proportional to the arc of the slot and consequently to the fitness of the corresponding individual.

Crossover produces two new offspring from two individuals by exchanging their substrings. The two most common forms of crossover in GAs are called the one-point and two-point crossover techniques. In one-point crossover, a position in the chromosome

is selected at random and the parts of two parents' chromosomes after the crossover position are exchanged. For example, the strings (10000100) and (11111111) could be crossed over after the third locus in each to produce the two offspring (10011111) and (11100100). In two-point crossover, two positions are chosen at random and the segments between them are exchanged.

Mutation allows the introduction of new genetic material by slightly changing the genotype of an individual. For example, the string (00000100) might be mutated in its second position to yield (01000100). Mutation is important for local search and helps prevent the GA from getting stuck at a local optimum. The common mutation operator flips each bit independently with a (small) probability μ, called the mutation rate. μ must be selected to be at a low level because otherwise mutation would randomly change too many *alleles* (bits) and the new individual would have nothing in common with its parent.

3.3 Fitness Function

The *fitness function* (i.e., heuristic function, objective function, penalty function, cost function, etc.) is used to measure the quality of a chromosome within a solution space. Fitness calculation of each chromosome in a population is the most time consuming process in a GA. Hence, it must be defined carefully to avoid unnecessary calculation. The relation between the fitness function and the chromosome representation, however, is benign and responsible for the immediate success of the algorithm. The

effectiveness of a fitness function in a GA is measured by the separation quality of fitness result for each chromosome in a search space Ω.

3.4 An Example

Now we give an example to explain how the GA works. A trivial problem might be to maximize a function, $f(x)$, where

$$f(x) = x^2, \text{ for integer } x \text{ and } 0 \leq x \leq 931 \tag{3.1}$$

Let us consider strings of length 5 in the search space Ω_5 , which we identify with the binary set $0, 1, \ldots, 31$ where, in an obvious way, **0** and **31** represents chromosomes (00000) and (11111), respectively. We start with the following initial random population $P(0)$, where for each $k \in P(0)$ we also calculate the corresponding fitness value $f(k)$:

$P(0)$	$f(k)$
01001	81
01010	100
00110	36
10011	361
00001	1
11000	576
10111	529
11001	625
11010	676
00110	36

Note that the maximum value obtained within $P(0)$ is 676 and that the average is 301.

We proceed to construct the next generation $P(1)$ by first selecting an intermediate population $P'(0)$ through the proportionate selection and then applying successively crossover for each pair of randomly selected parents and mutation to their offspring. In the table below we indicate by | the randomly selected crossover site and by underscore the bits where mutation has been applied:

$P'(0)$	$P(1)$	$f(k)$
101|11	10100	400
110|00	11011	729
01|010	01001	81
01|001	01010	100
11|000	11001	625
10|001	10000	256
110|10	11001	625
110|01	1101$\underline{1}$	729
1|1010	11010	676
1|1010	11010	676

Note that the population now has maximal value 729, the maximum of f, and average fitness 490.

Iterating this procedure, we obtain the populations for the first four generations as follows:

$P'(1)$	$P(2)$	$f(k)$	$P'(2)$	$P(3)$	$f(k)$	$P'(3)$	$P(4)$	$f(k)$
110|10	11010	676	110|11	11011	900	1111|0	11111	961
110|10	11010	676	111|11	11111	961	1111|1	11110	900
1101|0	11011	729	110|11	11011	729	11|111	11111	961
1101|1	11010	676	110|11	11011	729	11|111	11111	961
110|11	11010	676	1|1111	11$\underline{0}$11	729	1101|0	11010	676
110|10	11011	729	1|1111	11111	961	1111|0	11110	900
1|1011	11011	729	1101|0	11011	729	11|110	11111	961
1|1011	11$\underline{1}$11	961	1101|1	11010	676	11|111	11$\underline{0}$10	676
1101|1	11011	729	11|011	11111	961	1|1111	11111	961
1101|1	11011	729	11|111	11011	729	1|1111	11111	961

If we look at the evolution of the maximum and the average through these successive

generations, we find the following values for the maximum value in the population, its multiplicity (i.e., the number of copies of the best chromosome in a given population) and the average fitness values:

population	max	mult	avg
$P(0)$	676	1	301
$P(1)$	729	2	490
$P(2)$	961	1	731
$P(3)$	961	3	810
$P(4)$	961	6	892

It appears that the string (11111) which corresponds to the maximum of f quickly starts to dominate the population. For example, we may use the fact that a certain string accounts for more than a given percent of the chromosomes in a population as a stopping criterion. We also observe that the average fitness of the population gradually increases through successive generations.

This example illustrates that the GA does indeed function well, and that each successive generation tends to get better (i.e., better fitness values are observed in chromosomes in the population). As the average fitness value in a population increases, the population contains an increasing number of strings which are close to realizing the optimum.

Chapter 4

GA-based Topology Control Mechanisms

4.1 Introduction

The principal of *survival of the fittest* is the starting point of GAs which are typically applied to problems where deterministic or heuristic methods are not present or cannot provide satisfactory results. In this chapter, we describe our GA-based topology control mechanisms for spreading autonomous mobile nodes uniformly over a geographical area [48–50]. In our approach, Wireless mobile nodes adjust their speed and directions using a GA, where each mobile node exchanges its genetic information of speed and direction encoded in its chromosomes with its neighboring nodes.

Many applications, such as search and rescue missions, mine-field clearing, and self-

spreading of assets under harsh and bandwidth limited conditions, utilize uniform distribution of autonomous mobile nodes controlled by active running software agents over an unknown geographical area in MANETs. Typically, at each autonomous mobile node in a MANET, a software agent runs at the application layer; the agent can adjust the mobile node's direction and speed by only using the local information available from its neighbors.

The main objective of our GA-based topology control algorithm used by each mobile node is to converge toward a uniform distribution in an unknown geographical area and to reconfigure the team if any member becomes unavailable due to malfunctions or hostile activities. The GA is used to generate a solution set which includes a mobile node's direction and speed in the next time unit. For example, a group of mobile agents equipped with video cameras could be sent into a disaster site, where the agents, by running our GA-based topology control algorithm uniformly deploy themselves to cover the terrain. As they provide video transmission, some of the agents may be physically blocked by debris or may fail to operate, in which case the other mobile agents will re-arrange themselves to uniformly cover the area.

A second objective is to have a fully connected network. This objective is an important requirement for MANETs since it provides a multi-hop communication capability between any two nodes in the network. The network is defined to be *fully connected* if all nodes in the network are reachable by other nodes through either one-hop or multi-hop communication. Furthermore, this communication capability between any two mobile agents is maintained while the nodes are mobile.

The third objective is to maximize the area occupied by the mobile agents while minimizing the overlap of the communication coverage of the nodes. The third objective is to provide an optimum number of neighbors for each node depending on the network density. The second and third objectives result in better utility of network resources and less communication overhead due to the use of the minimum required number of mobile nodes.

4.2 Mobility Model

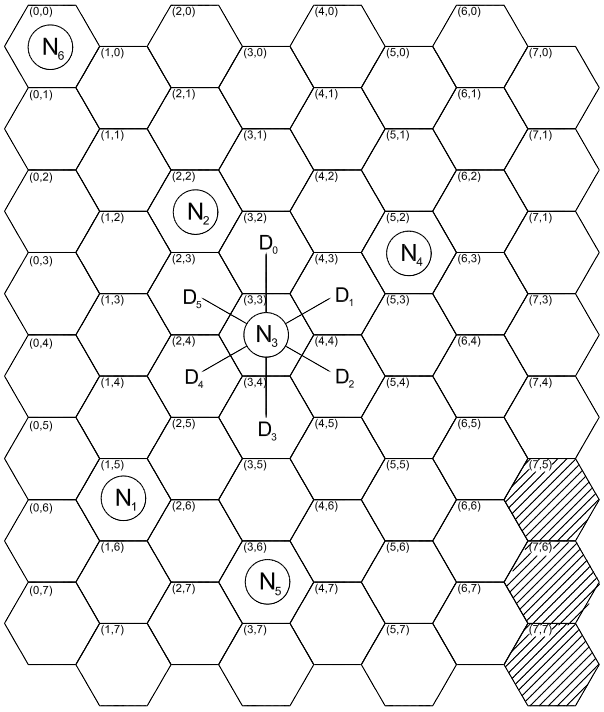


Figure 4.1: Six nodes distributed within an 8×8 hexagonal area partitioned into logical cells for $R_{com} = 3$.

We have adopted the mobility model introduced by Hökelek et al. [1,31] to represent the movement of mobile nodes, and how their location and velocity change over time.

We define a geographical terrain as a two dimensional $d_{max} \times d_{max}$ area divided into logical hexagonal cells, where a unique Cartesian coordinate pair (x,y) is assigned to each cell. Fig. 4.1 shows an area with 64 logical cells and seven mobile nodes; each node can move into six different directions (D_0 through D_5). A wireless link between two mobile nodes is represented by a vector whose dimensions are expressed in terms of layers. One layer is equal to the center-to-center distance between two neighboring cells. In general, for a mobile node in location (0,0) and another mobile node in location (x, y), the link state between these nodes is $< x, y >$ (i.e., $< x - 0, y - 0 >$). For example, in Fig. 4.1, for a mobile node N_3 in location (3,3) and another mobile node N_1 in location (1,5), the vector representing the wireless link between these nodes is <2,-2>.

Let R be the center-to-center distance between two neighboring cells. A wireless link between two mobile stations <x,y>, where x and y are integers ($0 \leq x, y \leq d_{max}$), is called *available* if two nodes are communicating with each other, otherwise *unavailable*. A wireless link is available only if $R \leq R_{com}$, where R_{com} is a positive integer representing the communication range. In Fig. 4.1, N_3 communicates with N_1, N_2, N_4, and N_5 if $R_{com} = 3$. The number of available links of a node is called its *degree* (σ). For example, the degree of N_3 is 4 for $R_{com} = 3$. The cells that are not within the transmission range of any MANET node are colored in gray.

A wireless link with the state of $< x, y >$ ($0 < x, y < d_{max}$) between two mobile nodes is called *available* if two nodes are communicating with each other; otherwise the link is said to be *unavailable*. For a wireless link $< x, y >$ connecting two mobile

Table 4.1: The probability distribution for a wireless link to switch from state $< x, y >$ to state $< x', y' >$ taken from [1].

x',y'	x,y	x-1,y	x-1,y-1	x,y-2	x+1,y-2	x+1,y-1	x+1,y	x,y-1	x+2,y-2	x+2,y-1
Probability	6/36	2/36	2/36	1/36	2/36	2/36	2/36	2/36	1/36	2/36

x',y'	x+1,y+1	x,y+1	x+1,y	x,y+2	x-1,y+2	x-1,y+1	x-1,y+2	x-2,y+1	x-2,y
Probability	2/36	2/36	1/36	1/36	2/36	2/36	1/36	2/36	1/36

nodes, after one time unit, each mobile node moves into one of its six neighbor cells with a probability of 1/6 for each direction. There are 36 possible next link states and, as some of them will result in the same vector, only 19 possible combinations as shown in Table 4.1. The direction and speed of movement for all nodes in the network are determined by our GA-based topology control algorithm running on each node. In our mobility model, a mobile node is not allowed to move beyond the area boundaries. For example, the mobile node on cell (0,0) can only move into one of the two different directions, namely D_2 and D_3. Suppose node N_3 in Fig. 4.1 wants to optimize its position at time t. Fig. 4.2 shows a possible location of N_3 after running our GA-based topology control algorithm at time $t + 1$.

4.3 Force-based Topology Control Algorithm

Our *Force-based Topology Control Algorithm* (FGA) uses a fitness function inspired by the force-based distribution in physics where each molecule attempts to remain in a balanced position and to spend minimum energy to protect its own position [25,46]. In our approach, FGA is run by each mobile node as a stand alone software agent. A virtual force is assumed to be applied by the neighboring nodes to any given node. At the equilibrium, the aggregate virtual force applied to a node by its neighbors

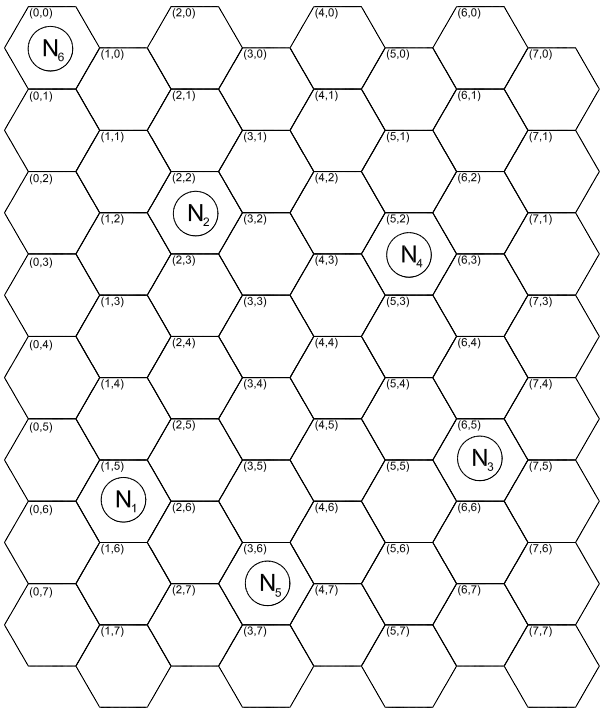


Figure 4.2: Expected location of N_3 after running our GA-based topology control algorithm at $(t+1)$ for $R_{com} = 3$.

should sum to zero. If the virtual force is not zero, our agent uses this non-zero virtual force value in its fitness calculation to adjust the nodes speed and direction of movement such that the total virtual force on the mobile node will be minimized. The value of this virtual force depends on the number of neighboring nodes within its communication range of R_{com} and the distance among them [51]. In FGA, a smaller fitness value indicates a better position for the corresponding node. The following characteristics make FGA a suitable candidate for MANET topology control applications: (*i*) the nodes are not required to be pre-distributed, (*ii*) the nodes have only the local neighborhood information to adjust their speed and directions, (*iii*) there is no central controller, and (*iv*) it runs as a software agent in each autonomous node.

In our framework, each node maintains a neighborhood information table to keep

records for its neighboring mobile nodes. A mobile node N_i runs its GA-based software agent every ΔT time units to find a better location (i.e., a location to result in a better fitness value for the node) to move (if exists) based on the information from its local neighborhood table. If the node cannot find a location to improve its fitness, it stops moving momentarily. More details of the fitness function used in our FGA are presented in Section 4.3.2.

Algorithm 4.1 Pseudo-code of our FGA.

```
while ! (stopCrit) do
  for all neighbors do
    if Neighbor is in the neighborhood table then
      Update the neighbor's information
    else
      Add the neighbor's information into table
    end if
  end for
  g ← 0 (generation counter)
  Initialize population P(0)
  Evaluate population P(0)
  while ! (evolveDone) do
    g ← g + 1
    Select P(g) from P(g-1)
    Crossover P(g)
    Mutate P(g)
    Evaluate P(g)
    if LocationFound then
      evolveDone := true
    end if
  end while
  if betterLocationFound then
    Move to new location
  else
    Wait (ΔT)
    Update neighborhood table
  end if
  Update stopCrit
end while
```

A pseudo code describing FGA is show in Algorithm 4.1. An initial population $P(0)$

containing N individuals is randomly generated. Initially the neighborhood information table of a given node is updated by the information received from the neighbors in its communication range. Each individual in $P(0)$ represents a candidate speed and movement direction for the node. All chromosomes in $P(0)$ are evaluated using a fitness function and sorted based on their fitness values. As mentioned previously, FGA is represented as a minimization problem and hence individuals are sorted in descending order of their fitness values. Then FGA proceeds with choosing pairs of chromosomes using the roulette wheel principle where a chromosome's probability of being selected is proportional to its fitness value. FGA applies crossover and mutation operators on the current population and obtains a new population of $P(1)$. FGA then repeats this procedure to obtain successive populations of $P(t)$, $t > 0$, until a termination criterion is satisfied (e.g., convergence tolerance of the best individuals reaches a certain limit, fitness value becomes below a predetermined value, or the number of generations exceeds a limit). If a node running FGA finds a better speed and movement direction that minimize the total virtual force on its current position, it adopts this new speed and direction; otherwise, it stops. A node continues running FGA until the condition called *stopCrit* is satisfied (e.g., the node obtains an acceptable level of uniformity in its vicinity).

4.3.1 Chromosomes in FGA

In the mobility model described in the previous section, each mobile node can move into one of six hexagonal directions in the area of interest (A_{ROI}). As an example,

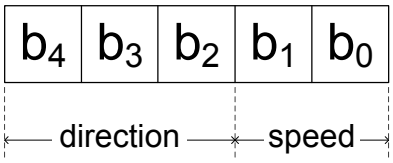


Figure 4.3: 5-bit chromosome.

let us assume that mobile nodes can move at four different speeds. Therefore, six different directions with four different speeds can be coded into a 5-bit chromosome ($< b_4b_3b_2b_1b_0 >$) as shown in Fig.4.3. The first three bits ($< b_4b_3b_2XX >$) represent hexagonal movement directions ($< 000 >$ representing north, $< 001 >$ northeast, $< 010 >$ southeast, $< 011 >$ south, $< 100 >$ southwest, and $< 101 >$ northwest). The last two bits of the chromosome ($< XXXb_1b_0 >$) are used for defining different speed values ($< 00 >$ for immobile, $< 01 >$ slower speed, $< 10 >$ normal speed, and $< 11 >$ faster speed). The speed implies the number of hexagonal cells that a node can move in one time unit. For example, if our FGA evolves to a chromosome value of $< 01110 >$, the corresponding mobile node should move three positions with normal speed heading south.

4.3.2 Fitness Function for FGA

A fitness function is used to measure the quality of the chromosomes within a solution space Ω. FGA's fitness function is based on the virtual forces applied to a node by its neighboring nodes. The force between two nodes depends on the distance between them and the number of other nodes within their communication ranges (i.e., the force from a closer neighbor is greater than a farther one). In our FGA implementation, a

mobile node uses the total force applied to it by the neighboring nodes located in its communication range R_{com} to decide its next direction and speed. The force exerted on node N_i by its neighboring node N_j is calculated as:

$$F_{ij} = \begin{cases} F_{max} & \text{if } d_{ij} = 0 \\ \sigma_i \left(d_{th} - d_{ij}\right) & \text{if } 0 < d_{ij} < d_{th} \\ 0 & \text{if } d_{th} \leq d_{ij} \leq R_{com} \end{cases} \tag{4.1}$$

where, d_{ij} is the Euclidean distance between nodes N_i and N_j, d_{th} is the threshold to define the local neighborhood, and σ_i is the degree for node N_i.

The force value is used as a part of each node's fitness calculation in our distributed FGA. The fitness function f_i is given as the sum of all the partial forces that k neighboring nodes exert on node N_i:

$$f_i = \sum_{j=1}^{k} F_{ij} = \sum_{j=1}^{k} \sigma_i \left(d_{th} - d_{ij}\right) \text{ for } 0 < d_{ij} \leq d_{th} \tag{4.2}$$

In FGA, the maximum distance that a mobile node can move in one time step is R_{com} units.

Notice that the goal of FGA is to find a set of parameter values (i.e., chromosomes), such as speed and direction, that minimizes the fitness function f_i in Eq. 4.2. The best fitness value (i.e., the lowest value of force) between nodes N_i and N_j is obtained

when the two nodes are d_{th} units apart from each other. Similarly, the worst fitness value corresponds to two nodes that are next to each other (i.e., $d_{ij} \approx 0$). A smaller fitness shows a better position for a mobile node.

In our decentralized FGA, a mobile node gathers information about its neighboring nodes' speed, direction, and location. Using the fitness function defined in Eq. (4.2), it generates candidate locations to move in the next time unit. These candidate solutions are ordered according to their fitness values from the lowest to the highest values. The lowest fitness value corresponds to the solution representing the least amount of force applied to the node, and, hence, the best one among the candidate solutions for that generation.

Theorem 1: *Let* $\sum_{j=1}^{k} {d_{ij}}^{t+1} \geq \sum_{j=1}^{k} {d_{ij}}^{t}$ *for* $0 < d_{ij} \leq d_{th}$, *and* k *neighbors for* N_i, *then* ${f_i}^{t+1} \leq {f_i}^{t}$.

Proof: From Eqs. 4.1 and 4.2, the force applied to node N_i by its k neighbors is defined as

$$\begin{aligned} {f_i}^{t} &= \sum_{j=1}^{k} \sigma_i \left(d_{th} - {d_{ij}}^{t} \right) \\ &= \sigma_i \left(k \cdot d_{th} - \sum_{j=1}^{k} {d_{ij}}^{t} \right) \end{aligned} \tag{4.3}$$

Similarly,

$${f_i}^{t+1} = \sigma_i \left(k \cdot d_{th} - \sum_{j=1}^{k} {d_{ij}}^{t+1} \right) \tag{4.4}$$

Since $\sum_{j=1}^{k} {d_{ij}}^{t+1} \geq \sum_{j=1}^{k} {d_{ij}}^{t}$, from Eqs. 4.3 and 4.4, we have ${f_i}^{t+1} \leq {f_i}^{t}$ ■

Lemma 1: FGA *does not let a node N_i move to a location that does not improve the node fitness.*

Proof: Our algorithm guarantees that N_i will not move to a new location where ${f_i}^{t+1} > {f_i}^{t}$. If the node fitness cannot be improved, N_i will remain in its current position as can be seen in Algorithm 4.1. ■

Lemma 2: *There exists k neighboring nodes, called stable cluster SC_α, if the sum of all k cluster node fitnesses is $\sum_{j=1}^{k} {f_j}^{t} = 0$.*

Proof: The fitness value for a node is defined as a positive number in Eq. 4.2. Therefore, if $\sum_{j=1}^{k} {f_j}^{t} = 0$, it implies that the fitness value for each of the k neighboring nodes is zero. From Eq. 4.2, $f_i = 0$ requires that node N_i should maintain a separation distance of $d_{th} \leq d_{ij} \leq R_{com}$ from each stable cluster node $N_j \in SC_\alpha$. ■

Lemma 3: *A stable cluster SC_α consisting of k nodes can become unstable when the fitness of a cluster node $N_i \in SC_\alpha$ changes such that ${f_i}^{t+1} > {f_i}^{t}$ due to a change in the number of nodes in SC_α.*

Proof: It is possible that the number of nodes change in SC_α to k' ($k \neq k$) when other nodes move into the vicinity of SC_α ($k' > k$) or when node(s) in SC_α malfunction ($k' < k$). In either of these cases, ${f_i}^{t+1} > {f_i}^{t}$ causes the cluster nodes to move to a new location to improve their fitnesses. ■

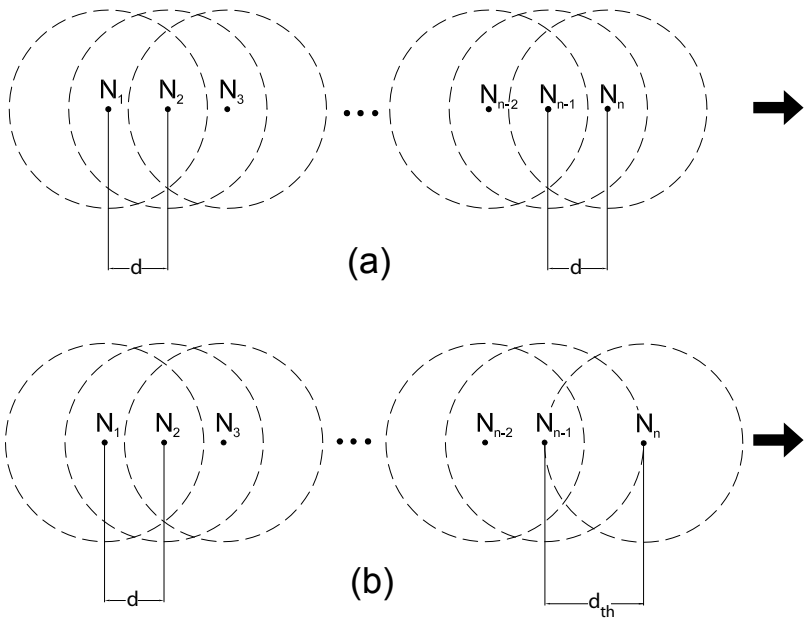


Figure 4.4: Congestion scenario of n nodes at steps (a) t, and (b) $t+1$.

Theorem 2: *If node N_i fitness does not improve such that ${f_i}^{t+1} > {f_i}^{t}$ (due to node congestion in its vicinity) at step $(t+1)$, node N_i will eventually improve its fitness in x time units ($x \in Z^+$) such that ${f_i}^{t+x} < {f_i}^{t}$ if A_{ROI} is large enough. Otherwise, max fitness is reached.*

Proof: Suppose we are given an A_{ROI} large enough to provide a uniform node distribution. For the worst case, suppose n nodes are placed within each other's close vicinity such that, without loss of generality, each node can only move in one direction (from left to right) as shown in Fig. 4.4. Based on Lemma 1, a node will not move unless it improves its current fitness. For simplicity, suppose the separation distance between two adjacent nodes of N_i and N_j $(i \neq j; i, j = 1, \ldots, n)$ is $0 < d_{ij} < d_{th}$ at t.

At step $t+1$, no node, except for N_n, can move. Node N_n and its neighboring node N_{n-1} are the only nodes that improve their fitnesses such that ${f_n}^{t+1} < {f_n}^{t}$ and

${f_{n-1}}^{t+1} < {f_{n-1}}^{t}$ since they get far apart from each other ($d_{n-1,n}^{t+1} = d_{th}$). Meanwhile, nodes N_1 through N_{n-2} remain in their same locations unable to improve their fitnesses.

At step $t+2$, Node N_{n-1} moves such as $d_{n-2,n-1}^{t+2} > d_{n-2,n-1}^{t+1}$ and hence N_{n-1} and N_{n-2} improve their fitnesses. At this point in time, node N_{n-2} needed to wait $t+2$ time units to lower its fitness such that ${f_{n-2}}^{t+2} < {f_{n-2}}^{t}$. Nodes N_1 through N_{n-3} are still unable to improve their fitnesses.

At step $t+x$, where $x \geq n-1$, node N_2 moves and consequently, node N_1 improves its fitness such as ${f_1}^{t+x} < {f_1}^{t}$. If N_2 cannot move, it implies that the other nodes are also immobile. In this case the outer level nodes reached the boundaries of A_{ROI} and cannot move any further to improve their fitnesses, which contradicts the main assumption of this theorem. ■

4.3.3 Performance Metrics

4.3.3.1 Normalized Area Coverage

Effectiveness in area coverage (called *normalized area coverage*, NAC) is an important performance metric for our GA-based topology control algorithm as it measures the success of mobile nodes' distribution over A_{ROI}. The goal of the FGA is to obtain the highest possible area coverage. NAC is defined as the ratio of the union of covered areas of each mobile node and the total terrain (areas with duplicated coverage by multiple nodes are not included). Each node covers the area within its communication

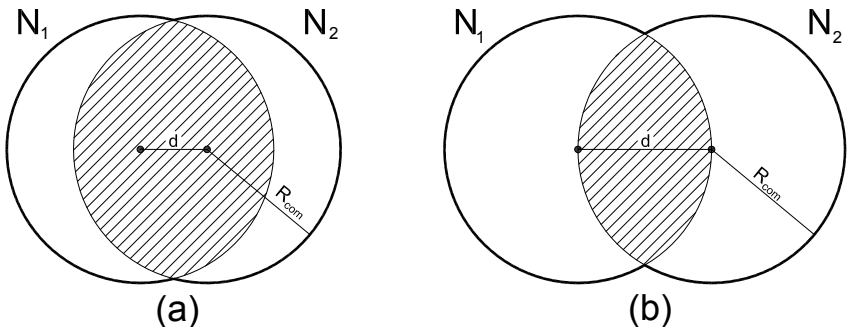


Figure 4.5: Intersections of two nodes. (a) $d = R_{com}/2$, (b) $d = R_{com}$.

range of R_{com}. NAC value is calculated as

$$\text{NAC} = \frac{\bigcup_{i=1,\cdots,N} A_i}{A} - A' \tag{4.5}$$

where A_i is the number of total hexagonal cells covered by mobile node N_i, A' is the area covered by obstacles (if any), and A is the total number of logical cells in the geographical terrain. NAC is a real number between 0 and 1, where 1 means that the area is fully covered by mobile nodes.

If a node is located well inside the geographical area, the full area of a circle around the node with a radius of R_{com} is counted as the covered region (i.e., πR_{com}^2) assuming that there are no other nodes overlapping with this node's coverage. If, however, a node is located near the boundaries of the geographical area, then only the partial area of the terrain covered by that node is included in NAC computation. For example, for R_{com}=3 in Fig. 4.1, NAC is equal to 61/64 since four cells are not covered by any node. For this objective, an agent located on the boundary only partially contributes to NAC. For example, in Fig. 4.1, N_6, which is located on the boundary cell, covers only 12 cells while N_3, which is in the middle of A_{ROI}, covers 37 cells.

$\sum d_{ij} = d$	0	2	4	6	8	10	20
$A_1 \cap A_2$	314.16	274.23	234.70	195.98	158.53	122.84	0
A_T	314.16	354.09	393.62	432.33	469.78	505.48	628.32

Table 4.2: Total and overlap areas of the two nodes in Fig. 4.5 for different values of d and $R_{com} = 10$.

Figs. 4.5 (a) and (b) show two nodes with separation distances of $d = R_{com}/2$ and $d = R_{com}$, respectively. Let A_1 and A_2 be the covered areas of the two nodes shown in Fig. 4.5. A_T can be expressed as:

$$A_T = \bigcup_{i=1}^{n} A_i = A_1 + A_2 - (A_1 \cap A_2) \tag{4.6}$$

For example, assuming a circular area for A_1 and A_2, A_T in Fig. 4.5 is given by:

$$A_T = 2\pi R_{com}^2 - 2R_{com}^2 Cos^{-1}\left(\frac{d}{2R_{com}}\right) - \frac{d}{2}\sqrt{4R_{com}^2 - d^2} \tag{4.7}$$

In Table 4.2, one can notice that A_T gets larger and the overlapping region becomes smaller as the separation distance d increases. The largest value for A_T is when $d = 20$, which is twice the value of R_{com}. However this separation distance is infeasible because the two nodes are not able to communicate with each other. In our analysis, the allowed maximum separation distances are such that $d \leq R_{com}$.

It is possible to use Eq. 4.6 to find a similar relationship for three nodes. Let A_1,A_2, and A_3 be the covered areas by three nodes, then A_T is given by:

$$
\begin{aligned}
A_T &= A_1 + A_2 - (A_1 \cap A_2) + A_3 \\
&\quad -[(A_1 \cap A_3) + (A_2 \cap A_3) - (A_1 \cap A_2 \cap A_3)] \\
&= A_1 + A_2 + A_3 - [(A_1 \cap A_2) + (A_1 \cap A_3) \\
&\quad +(A_2 \cap A_3)] + (A_1 \cap A_2 \cap A_3) \qquad (4.8)
\end{aligned}
$$

In order to express Eq. 4.8 more conveniently, we set

$$
\begin{aligned}
S_1 &= A_1 + A_2 + A_3 \qquad (4.9) \\
S_2 &= A_1 \cap A_2 + A_1 \cap A_3 + A_2 \cap A_3, \text{ and} \qquad (4.10) \\
S_3 &= A_1 \cap A_2 \cap A_3 \qquad (4.11)
\end{aligned}
$$

Then we have

$$
A_T = S_1 - S_2 + S_3 \qquad (4.12)
$$

There is a generalization of Eq. 4.12 that is valid for n nodes. Let $A_1, \ldots, A_n$ be the covered areas for each node. Let us consider n nodes S_r $(1 \leq r \leq n)$ as follows

$$
S_r = \sum_{1 \leq i_1 < \cdots < i_r \leq n} A_{i_1} \cap \cdots \cap A_{i_r} \qquad (4.13)
$$

Then we have

$$S_1 = A_1 + \cdots + A_n \tag{4.14}$$

$$S_2 = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} A_i \cap A_j, \text{ and} \tag{4.15}$$

$$S_n = A_1 \cap \cdots \cap A_n \tag{4.16}$$

The desired equation for $\bigcup_{i=1}^{n} A_i$ is given by:

$$\begin{aligned} A_T &= \bigcup_{i=1}^{n} A_i = \sum_{r=1}^{n} (-1)^{r-1} S_r \\ &= S_1 - S_2 + \cdots + (-1)^{n-1} S_n \end{aligned} \tag{4.17}$$

Now, let us show that the increase in the sum of the separation distances among the nodes increase implies the improved NAC value.

Lemma 4: *Let* $\sum_{j=1}^{k} d_{ij}^{\,t+1} \geq \sum_{j=1}^{k} d_{ij}^{\,t}$ *for* $0 < d_{ij} \leq d_{th}$, *and* k *neighbors for* N_i; *then* NAC $^{t+1} \geq$ NAC ^{t}.

Proof: NAC is directly proportional to the union of covered area of each node ($\bigcup_{i=1}^{n} A_i$) as shown in Eq. 4.5. Therefore, larger values of covered area of each node increases NAC value. To get higher values of NAC, the overlap areas between nodes (e.g., S_2 in Eq. 4.16) need to be minimized, implying that the separation distance between nodes should increase. ■

4.3.3.2 Deployment Time

Deployment time is a metric that represents the total time it takes for the mobile nodes to converge toward a uniform distribution over a geographical terrain. The deployment time includes the GA processing time, communication overhead and the time for physically moving the mobile nodes into the desired locations at each iteration of FGA. This metric is used to analyze two important parameters of FGA. The first parameter corresponds to the amount of time the mobile nodes have to spend spreading over an area. The second parameter represents the speed of area coverage recovery in the case of lost nodes due to equipment malfunctions or hostile activities.

4.4 Mean-node Degree Topology Control Algorithm

In early stages of our research, we adapted the analytical mean node degree calculation from [1, 52] to construct the appropriate fitness function for our GA-based topology control algorithm called *Mean-node Degree Topology Control Algorithm* (MDGA). In Ref. [52], the mean node degree ($\overline{N}$) is analytically derived as an approximation of the optimum number of neighbors yielding to an uniform node distribution in MANET. $\overline{N}$ depends on the geographic area size, the communication range (R_{com}), and the number of mobile nodes in the network. $\overline{N}$ plays a crucial role in determining the convergence of MDGA and we conjecture that the near-optimal agent deployment can be achieved if a mobile node maintains its node degree approximately around $\overline{N}$.

We studied the effectiveness of MDGA and compared them to random-walk and Hill Climbing approaches. Three different sets of experiments where conducted, where mobile agents constantly move using the directions and speeds determined by the GA (Case 1), stop intermittently once a perfect fitness is obtained until other agents move into their vicinity (Case 2), and stop permanently with their close neighbors once a perfect fitness is obtained (Case 3). We also developed a Java-based simulation software to study the performance of these cases with respect to NAC, average fitness values, average traveled time, and convergence.

4.4.1 Fitness Function for MDGA

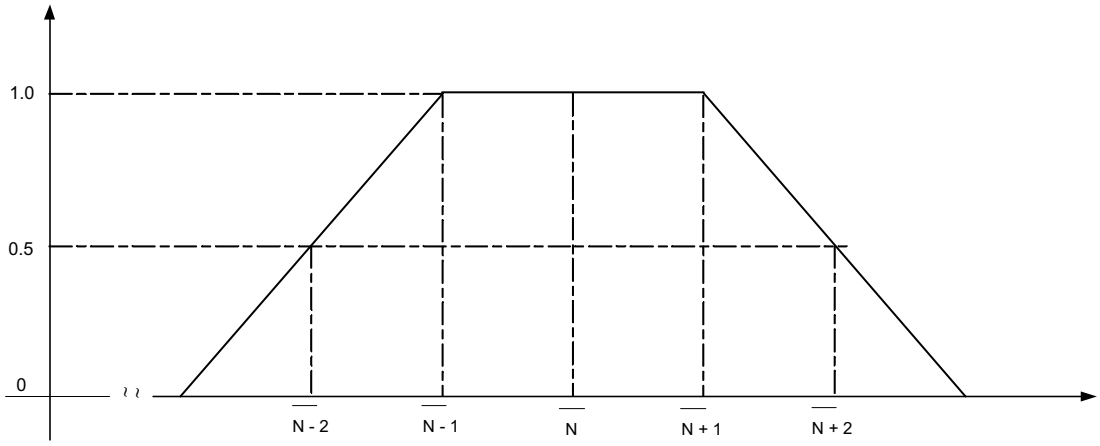


Figure 4.6: Fitness function with step distribution for MDGA algorithm.

The fitness function used in MDGA is designed such that, for a node N_i, it yields the highest fitness value if the number of neighbors σ (degree) for node N_i is within the range of $x < \sigma < y$, where x and y are integers. The agent nodes whose degrees are not in this range will have lower fitness values. x and y are set to the values around the mean node degree, which is calculated analytically. The generalized step

Table 4.3: Numerical results of $\overline{N}$ for different sets of N and R_{com} (n_{tot}=50).

N	60	80	100	120	150	60	80	100	120	150
R_{com}	10	10	10	10	10	12	12	12	12	12
$\overline{N}$	2.59	3.48	4.36	5.24	6.56	3.68	4.93	6.18	7.42	9.30

function used as the fitness function in our study is shown in Fig. 4.6 where the node degree yielding the best fitness is represented as $\overline{N}$. Since the optimum node degree changes with the agents' locations relative to the area boundaries, our GA fitness function uses this generalized function as an approximation. In this fitness function, if the number of neighbors of a node is between $\overline{N}-1$ and $\overline{N}+1$, the node's fitness is honored with a 1.0. If the node degree is either $\overline{N}-2$ or $\overline{N}+2$, the fitness is set to 0.5. A mobile agent has a fitness of 0.0 when its number of neighbors is either less than $\overline{N}-2$ or greater than $\overline{N}+2$. For example, suppose there are 100 mobile agents in a hexagonal lattice, and the communication range, R_{com}, of any agent is 10 layers. In this case, the mean node degree, $\overline{N}$, is computed as 4.36. Hence, the fitness of a mobile agent will result in 1.0 when it has either 3, 4, or 5 neighbors; if the node has either 2 or 6 neighbors, its fitness will be 0.5. For other values of the number of neighbors, the node fitness will be 0.0.

Table 4.3 shows the calculated numerical value of $\overline{N}$ for different network densities [1, 52]. $\overline{N}$, is used for assigning the fitness function values for the related number of nodes and communication range.

4.4.2 Tournament

As mention in Chapter 3, the tournament chooses two genotypes from the gene pool, compares them, and then permits the one with the better fitness to reproduce. In other words, by using the *elitist* policy, the best individuals are selected for the reproduction of the new generation. In MDGA, each mobile agent randomly selects a neighbor to perform the tournament, where the paired agents exchange their fitness information with each other to decide the winner agent of the tournament.

We studied different approaches to perform the tournament mechanism for the knowledge sharing bio-inspired agents to obtain a uniform distribution of the nodes over a geographical terrain. All cases utilize the same fitness function described in Section 4.4.1; however, each case has a different speed adjustment procedure as described below.

Case 1: Two mobile agents that have been selected as a couple exchange their genetic material. The mobile agent with the lower fitness value adapts the chromosomes of the fitter agent (i.e., the losers adapt the speed and the direction of the winners) [53].

Case 2: One may argue that if a mobile agent reaches a higher fitness value, it should keep its current position and hence should decrease its speed. Adversely, the nodes whose fitness values are low should increase their speeds to move into areas that will yield better fitness values. We introduce Case 2 to evaluate effectiveness of such a mechanism in terms of improving the mobile agent distribution. In Case 2, a pair of mobile agents that are selected for tournament adjust their speeds and directions

based on the genetic material that they exchange with each other. However, the agents with better fitness slow down while the ones with lower fitness speed up. If a mobile agent's fitness value eventually reaches to value of one (i.e., the highest fitness value), it stops moving. A stopped node may start moving again later if either other nodes come into its vicinity or its current neighbors move away from it, which causes the node's fitness value to be less than one (recall that if a mobile agent has too many or too few neighbors, its fitness value will be below one as shown in the fitness function in Fig. 4.6).

Case 3: In this case we explore a different approach such that the nodes in relatively better positions influence the behavior of their neighbors to obtain a uniform distribution of the mobile agents into a geographical terrain. In Case 3, a nodes is classified as either a `leader`, `follower`, or a `non-follower`, depending on its current relative position with respect to its neighboring agents. A node is considered a `leader` if it has the optimum number of neighbors (i.e., its node degree is equal to the mean node degree $\overline{N}$). A node becomes a `follower` if one of its neighbors is a `leader`. The nodes that are not classified as either `leaders` or `followers` are labeled as `non-followers`. At any given time, during the tournament, a `non-follower` mobile agent becomes a `leader` if its fitness value is one (i.e., its number of neighbors is equal to $\overline{N}$), none of its neighbors is a `leader` (two `leaders` cannot be neighbors) and it does not have any `follower` neighbors (`leaders` do not share any `followers`). If a node satisfies these conditions, it calls itself a `leader` and notifies its neighbors that they are now to become its `followers`. Once a node becomes a `leader` and its neighbors `followers`,

they form a so-called cluster and stop moving. The `followers` then adjust their relative positions around the `leader` so that they are from equidistant from the `leader`. Once a cluster is formed, its members never move again.

4.4.3 Crossover and Mutation

The crossover operator produces two offspring chromosomes for each generation. For Case 1 experiments, each mobile node selects a mate different than the tournament mate to perform the crossover. Then each couple decides on a random crossover point and exchanges genetic information based on that crossover point. As a result, each couple will have an offspring heavily influenced by the winners of the tournament for this generation as explained in Chapter 3. Based on the crossover operation, each mobile node will have a new speed and direction. In the experiments for Case 2, all but the stopped nodes participate in the crossover operation as described for Case 1. However, a stopped node may also participate in the crossover if, at some point in the future, its fitness value becomes less than one due to the nodes moving into or out of its vicinity. In Case 3, however, since the stopped nodes of each cluster, one leader and its followers, never move again, they do not participate in the crossover operation once they stop moving.

For mutation, almost any operation that changes the order of genes within a chromosome or that changes a gene's value (e.g., its location or direction) is a valid mutation operator. The chosen mutation operator iterates over all genes and mutates each with

some low probability of mutation (μ). The stopped nodes of Cases 2 and 3, which do not participate in the crossover, will not perform mutation either (See Chapter 3).

Chapter 5

Dynamical System Model

We apply Vose's *dynamical system model* [40] to study the convergence of our decentralized topology control mechanism (FGA) where a genetic algorithm (GA) is run by each autonomous mobile node to achieve a uniform spread over an unknown geographical area. The dynamical system model calculates the cumulative effects of GA operators of selection, mutation, and crossover as a population evolves through generations. This model provides expected population distribution after each generation by observing the changes in the chromosomes. Also, using this model, it is possible to analyze the effect of the initial population on the final convergence of our FGA.

Dynamical system model uses the information in the past populations to create the state of the current population. Due to the nondeterministic nature of the genetic operators, we cannot exactly predict what the next population will be. However, we

can calculate the probability distribution over the set of possible populations defined by the genetic information of the past populations (or *states*) to estimate the expected next population.

As the size of a population grows, the probability that the next state of a population will be the targeted one approaches to one since the chance that copies of the best chromosome will be included in the population increases. In this case, the GA behavior becomes less stochastic but more deterministic, and, hence, the trajectory of expected next population gives a relatively safe indication of the actual GA behavior.

5.1 Population Representation

Let Ω denote the search space, and n the cardinality of Ω. Then, using a fixed-length binary string representation, $\Omega = \{0,1\}^{\ell}$, where ℓ is the string length. We will identify the elements of Ω with the integers in the range of $[0, n)$.

A population P can be represented as an incidence vector: $P = \langle P_0, P_1, \ldots, P_{n-1} \rangle$, where P_k is the number of copies of individual $k \in \Omega$ in the population, hence $\sum_{k=0}^{n-1} P_k = N$, where N is the population size. To obtain a more general representation, we can also describe the population as a *population vector* $p = \langle p_0, p_1, \ldots, p_{n-1} \rangle$, in which p_k is the proportion that an individual $k \in \Omega$ appears in the population.

For example, suppose that a population consists of $\{10, 00, 01, 10, 11, 10, 00, 10, 10, 11\}$ for $\ell = 2$, ; therefore, $N = 10$ and $p_0 = 0.2$ since the individual 00 appears twice in

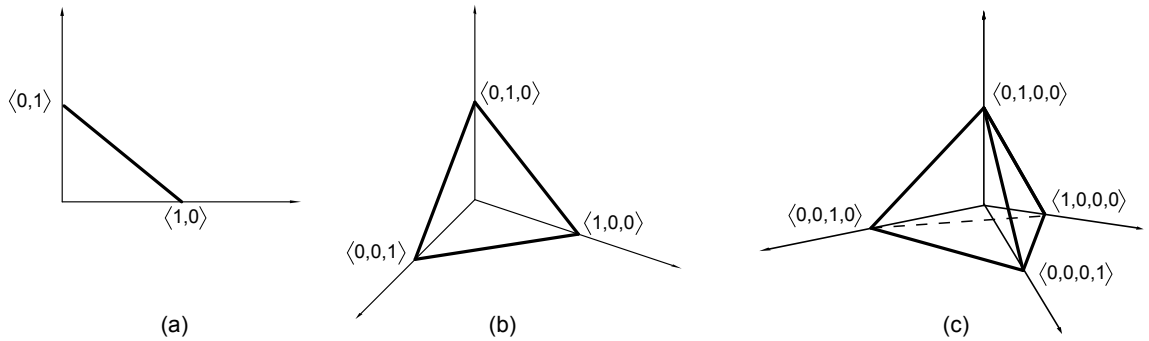


Figure 5.1: The simplex of possible populations when (a) $n = 2$, (b) $n = 3$, and (c) $n = 4$.

the population. For this example, $p = \langle 0.2, 0.1, 0.5, 0.2 \rangle$.

Properties of population vectors are:

- p is an element of the vector space R^n (addition and/or multiplication by scalar produce other vectors within R^n)
- Each entry p_k must lie in the range $[0, 1]$
- All entries of p sum to 1 ($\sum_{k=0}^{n-1} p_k = 1$)

The set of all vectors in R^n that satisfy these properties is called a `simplex` and denoted by Λ. A simplex can be seen as either the set of population-size-independent representations of populations, or the set of probability distributions over Ω. In general,

$$\Lambda = \left\{ p \in R^n : p_i \geq 0 \text{ and } \sum_{i=0}^{n-1} p_i = 1 \right\} \tag{5.1}$$

We can now view the actions of our FGA on a population as a trajectory of vectors $p \in R^n$. When $n = 2$, the simplex is a straight line segment in the plane R^2, running from

$\langle 1, 0\rangle$ to $\langle 0, 1\rangle$ as shown in Fig. 5.1(a). All real populations correspond to points within the simplex. However, since components in the corresponding population vectors must be rational numbers not all points in the simplex correspond to finite populations. Therefore, only those rational points with common denominator N correspond to valid population vectors. For example, vertex $\langle 1, 0\rangle$ represents populations with copies of individual 1 only. The points in between are populations containing a mixture of individuals. When $n = 3$ and $n = 4$, the simplex is a triangle embedded in R^3 (Fig. 5.1(b)) and a tetrahedron embedded in R^4 (Fig. 5.1(c)), respectively.

5.2 Discrete-time Dynamical System

FGA can be described through a heuristic function $\mathcal{G}(p) : \Lambda \rightarrow \Lambda$, where $\mathcal{G}(p)$ contains all the details of selection , crossover, and mutation operators represented by $\mathcal{F}$, $\mathcal{C}$, and $\mathcal{U}$ heuristic functions, respectively. In other words, $\mathcal{G}(p) = \mathcal{U}(\mathcal{C}(\mathcal{F}(p))) = \mathcal{U} \circ \mathcal{C} \circ \mathcal{F}(p)$ where $\mathcal{G}(p)$ is a discrete-time dynamical system that has the following interpretations:

- $\mathcal{G}(p)$ is the *expected* next generation population
- $\mathcal{G}(p)$ is the *limiting next population* as the population size goes to infinity
- $\mathcal{G}(p)_k$ is the *probability* that $k \in \Omega$ is selected to be in the next generation

Later sections describe each of the three heuristic functions in more detail.

5.3 Selection

Selection is the first operator in FGA. In order to construct our model, we first define *selection heuristic function* $\mathcal{F}(p) : \Lambda \to \Lambda$. Let $p = (p_0, p_1, ..., p_{n-1})$ be our current population and the probability $P_r(k)$ that any individual k will be selected for the next population is given by:

$$Pr(k) = \frac{f_k p_k}{\sum_{i=0}^{n-1} f_i p_i} \tag{5.2}$$

where f_i represents the fitness value of individual $i \in \Omega$. Then the fitness proportional selection operator $\mathcal{F}(p)$ expressed in terms of matrices as

$$\mathcal{F}(p) = \frac{diag(f)p}{f^T p} \tag{5.3}$$

where f is the fitness function expressed as a vector ($f_k = f(k)$), $diag(f)$ is the diagonal matrix with entries from vector f along its diagonal, and $f^T p$ is the average fitness of the population p (inner product of vectors f and p).

We now give an example which we will continue through the remaining sections.

Let $\ell = 2$, $N = 10$, initial population $P = \langle 2, 1, 5, 2 \rangle$, and fitness function $f(x) = x^2$. This gives $p = \langle \frac{1}{5}, \frac{1}{10}, \frac{1}{2}, \frac{1}{5} \rangle$, $f_0 = 0$, $f_1 = 1$, $f_2 = 4$, and $f_3 = 9$. Using Eq. (5.3), we calculate $F(p)$ as follows:

$$\mathcal{F}(p) = \frac{\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 9 \end{bmatrix} \begin{bmatrix} 0.2 \\ 0.1 \\ 0.5 \\ 0.2 \end{bmatrix}}{\begin{bmatrix} 0 & 1 & 4 & 9 \end{bmatrix} \begin{bmatrix} 0.2 \\ 0.1 \\ 0.5 \\ 0.2 \end{bmatrix}} = \frac{\begin{bmatrix} 0 \\ 0.1 \\ 2.0 \\ 1.8 \end{bmatrix}}{3.9} = \begin{bmatrix} 0 \\ 0.0256 \\ 0.5128 \\ 0.4615 \end{bmatrix} \tag{5.4}$$

If proportional selection is the only genetic operator, then $\mathcal{G} = \mathcal{F}$ and we now have the evolutionary equation for $\mathcal{G}$.

5.4 Crossover

Crossover operator never changes the number of zeros or ones in a given chromosome pair, but only shuffles their bits. In our algorithm, we use *one-point crossover* (1X) where a *crossover point* is chosen at random from the numbers $1, \ldots, \ell - 1$, and a new chromosome is produced by combining the pieces of the original parent chromosomes. For instance, if we combine parent chromosomes $a_0a_1a_2a_3$ and $b_0b_1b_2b_3$ with a crossover point of 2, the offspring chromosomes would be $a_0a_1b_2b_3$ and $b_0b_1a_2a_3$.

Crossover can be easily written as a binary string or *mask* when implementing crossover operator in a computer algorithm. Therefore, any linear crossover operator is represented by a vector $m \in \Omega$ so that the offspring of parents a and b are $a \otimes m \oplus b \otimes \overline{m}$ and $a \otimes \overline{m} \oplus b \otimes m$, where $\overline{m}$ is the complement of m, and $\otimes, \oplus$

denote component-wise multiplication and addition respectively. For example if we have parents (01011) and (10110), the mask (11100) indicates that we take the first three bits of the first parent and the last two from the second parent to produce (01010).

The effects of applying crossover can be represented with the *crossover heuristic function* $\mathcal{C}(p) : \Lambda \to \Lambda$ such that the kth component of $\mathcal{C}(p)$ is the probability that individual $k \in \Omega$ results from applying crossover to population p.

The probability that a given chromosome k is created by applying the crossover operator to a population p is found by summing over all the possible ways this can happen. Hence, $\mathcal{C}(p)$ is defined as:

$$\mathcal{C}(p)_k = \sum_{i,j} p_i p_j r(i,j,k) \tag{5.5}$$

where individuals $i, j \in \Omega$, and $r(i,j,k)$ is the probability of creating chromosome k from parent chromosomes i and j.

One-point crossover is not symmetric, so that $r(i,j,k) \neq r(j,i,k)$, however we can define symmetric matrices M_k having $(i,j)^{\text{th}}$ entries as:

$$M_k = \frac{1}{2}\left(r(i,j,k) + r(j,i,k)\right) \tag{5.6}$$

Therefore,

$$\mathcal{C}(p)_k = p^T M_k p \tag{5.7}$$

To continue the example started in the previous section, there are two valid masks $m \in \Omega$ for $\ell = 2$ that represent one-point crossover. They are $m = \mathbf{2} = (10)$ and $m = \mathbf{3} = (11)$ with probabilities μ_c and $(1 - \mu_c)$, respectively. We now calculate $r(i, j, 0)$, the probability that individuals i and j cross to form individual $\mathbf{0} = (00)$ with a crossover rate of μ_c using $m = \mathbf{2} = (10)$ and $m = \mathbf{3} = (11)$:

$$r(i, j, \mathbf{0}) = \begin{bmatrix} \mu_c + (1 - \mu_c) & \mu_c & 0 & 0 \\ 1 - \mu_c & 0 & 0 & 0 \\ \mu_c + (1 - \mu_c) & \mu_c & 0 & 0 \\ 1 - \mu_c & 0 & 0 & 0 \end{bmatrix} \tag{5.8}$$

For example, to produce $\mathbf{0} = (00)$ from parents $\mathbf{0} = (00)$ and $\mathbf{2} = (10)$, we can use either mask (10) in view of $(00) \otimes (10) \oplus (10) \otimes (01) = (00)$, or mask (11) in view of $(00) \otimes (11) \oplus (10) \otimes (00) = (00)$. The probability is clearly $\mu_c + (1 - \mu_c)$, since either the first bit must come from $\mathbf{0}$ and the other from $\mathbf{2}$ with probability μ_c, or both bits come from $\mathbf{0}$ with probability $(1 - \mu_c)$.

We now proceed to calculate $r(j, i, 0)$, the probability that individuals j and i cross to form individual $\mathbf{0} = (00)$ using $\overline{m} = \mathbf{1} = (01)$ and $\overline{m} = \mathbf{0} = (00)$:

$$r(j, i, \mathbf{0}) = \begin{bmatrix} \mu_c + (1 - \mu_c) & 1 - \mu_c & \mu_c + (1 - \mu_c) & 1 - \mu_c \\ \mu_c & 0 & \mu_c & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{5.9}$$

From the matrix above, we can see that the probability of producing $\mathbf{0} = (00)$ from parents $\mathbf{0} = (00)$ and $\mathbf{2} = (10)$ is zero since offspring $(00) \otimes (01) \oplus (10) \otimes (10) = (10)$

and $(00) \otimes (00) \oplus (10) \otimes (11) = (10)$ do not result in individual $\mathbf{0} = (00)$.

We now combine matrices $r(i, j, 0)$ and $r(j, i, 0)$ to obtain symmetric matrix M_0 using Eq.(5.6):

$$M_0 = \frac{1}{2}\left(r(i, j, 0) + r(j, i, 0)\right) \tag{5.10}$$

$$M_0 = \begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{2} & \frac{(1-\mu_c)}{2} \\ \frac{1}{2} & 0 & \frac{\mu_c}{2} & 0 \\ \frac{1}{2} & \frac{\mu_c}{2} & 0 & 0 \\ \frac{(1-\mu_c)}{2} & 0 & 0 & 0 \end{bmatrix} \tag{5.11}$$

And doing this for each individual $k \in \Omega$, we have

$$\mathcal{C}(p) = \left(p^T M_0 p, p^T M_1 p, \ldots, p^T M_{n-1} p\right) \tag{5.12}$$

In the same manner as M_0 was concluded, M_1 is calculated as

$$M_1 = \frac{1}{2}\left(r(i, j, 1) + r(j, i, 1)\right) \tag{5.13}$$

$$M_1 = \begin{bmatrix} 0 & \frac{1}{2} & 0 & \frac{\mu_c}{2} \\ \frac{1}{2} & 1 & \frac{(1-\mu_c)}{2} & \frac{1}{2} \\ 0 & \frac{(1-\mu_c)}{2} & 0 & 0 \\ \frac{\mu_c}{2} & \frac{1}{2} & 0 & 0 \end{bmatrix} \tag{5.14}$$

The probabilities in each matrix M_k are exactly those that are in M_0, but shuffled around according to the permutation matrix α_k. Therefore, we only have to calculate M_0 (*mixing matrix*) to capture the whole effect of crossover. This means that

$$M_k = \alpha_k M_0 \alpha_k^T \text{ for each } k \in \Omega \tag{5.15}$$

In order to calculate permutation matrix α_k, we consider each element within Ω as a group under bitwise addition (modulo 2). In our example, the group table for addition is:

	00	01	10	11
00	00	01	10	11
01	01	00	11	10
10	10	11	00	01
11	11	10	01	00

(5.16)

The element $\mathbf{0} = (00)$ is the identity in this group. The corresponding permutation matrices are

$$\alpha_0 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \alpha_1 = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

$$\alpha_2 = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \quad \alpha_3 = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \tag{5.17}$$

In general, we can observe that for any given α_k, a 1 is set where entry $(i, j)^{\text{th}}$ in Table 5.16 is equal to k. For instance, for α_2, a 1 is placed where entry $(i, j)^{\text{th}}$ is equal to $i \oplus j = \mathbf{2} = (10)$, otherwise a 0 is assigned.

To continue with our example, let us assume $\mu_c = 0.4$, hence the complete set of matrices for this crossover operator using Eq.(5.15) are:

$$M_0 = \begin{bmatrix} 1 & 0.5 & 0.5 & 0.3 \\ 0.5 & 0 & 0.2 & 0 \\ 0.5 & 0.2 & 0 & 0 \\ 0.3 & 0 & 0 & 0 \end{bmatrix} \quad M_1 = \begin{bmatrix} 0 & 0.5 & 0 & 0.2 \\ 0.5 & 1 & 0.3 & 0.5 \\ 0 & 0.3 & 0 & 0 \\ 0.2 & 0.5 & 0 & 0 \end{bmatrix}$$
$$M_2 = \begin{bmatrix} 0 & 0 & 0.5 & 0.2 \\ 0 & 0 & 0.3 & 0 \\ 0.5 & 0.3 & 1 & 0.5 \\ 0.2 & 0 & 0.5 & 0 \end{bmatrix} \quad M_3 = \begin{bmatrix} 0 & 0 & 0 & 0.3 \\ 0 & 0 & 0.2 & 0.5 \\ 0 & 0.2 & 0 & 0.5 \\ 0.3 & 0.5 & 0.5 & 1 \end{bmatrix} \tag{5.18}$$

Now, we calculate $\mathcal{C}(p)$ using Eq.(5.7)

$$\mathcal{C}(p)_k = \mathcal{F}(p)^T M_k \mathcal{F}(p) = \begin{bmatrix} 0.0053 \\ 0.0204 \\ 0.5076 \\ 0.4668 \end{bmatrix} \tag{5.19}$$

5.5 Mutation

The mutation operator randomly changes a bit with usually a small probability in order to provide diversity to a population and protect against getting stuck in a local optima. Mutation can be defined by means of mutation masks. If $j \in \Omega$, then the result of mutating j using a mutation mask $m \in \Omega$ is $j \oplus m$. The mutation heuristic is defined by giving a probability distribution $\mu \in \Lambda$ over mutation masks. In other words, μ_m is the probability that $m \in \Omega$ is used. Given a population $p \in \Lambda$, the *mutation heuristic* $\mathcal{U}(p) : \Lambda \to \Lambda$ is defined by

$$\mathcal{U}(p)_k = \sum_{j \in \Omega} \mu_{j \oplus k} p_j \tag{5.20}$$

where $\mu_{j \oplus k}$ is the probability that an individual j mutates to an individual k using a mutation mask $m \in \Omega$. The contribution of mutation can also be described in terms of a $n \times n$ matrix U called *mutation matrix* that directly gives the effect of mutation on a population vector. Hence, Eq.(5.20) can simply be shown as

$$\mathcal{U}(p)_k = Up \tag{5.21}$$

We continue the numerical example. Let us assume $\mu_m = 0.1$, then the mutation matrix U is

$$U = \begin{bmatrix} (1-\mu_m)^2 & (1-\mu_m)\mu_m & \mu_m(1-\mu_m) & \mu_m^2 \\ (1-\mu_m)\mu_m & (1-\mu_m)^2 & \mu_m^2 & \mu_m(1-\mu_m) \\ \mu_m(1-\mu_m) & \mu_m^2 & (1-\mu_m)^2 & (1-\mu_m)\mu_m \\ \mu_m^2 & \mu_m(1-\mu_m) & (1-\mu_m)\mu_m & (1-\mu_m)^2 \end{bmatrix} \tag{5.22}$$

$$U = \begin{bmatrix} 0.81 & 0.09 & 0.09 & 0.01 \\ 0.09 & 0.81 & 0.01 & 0.09 \\ 0.09 & 0.01 & 0.81 & 0.09 \\ 0.01 & 0.09 & 0.09 & 0.81 \end{bmatrix} \tag{5.23}$$

Now, we calculate $\mathcal{U}(p)$ using Eq.(5.21)

$$\mathcal{U}(p) = U\mathcal{C}(p) \tag{5.24}$$

$$\mathcal{U}(p) = \begin{bmatrix} 0.81 & 0.09 & 0.09 & 0.01 \\ 0.09 & 0.81 & 0.01 & 0.09 \\ 0.09 & 0.01 & 0.81 & 0.09 \\ 0.01 & 0.09 & 0.09 & 0.81 \end{bmatrix} \begin{bmatrix} 0.0082 \\ 0.0632 \\ 0.2775 \\ 0.6510 \end{bmatrix} = \begin{bmatrix} 0.0564 \\ 0.0641 \\ 0.4538 \\ 0.4257 \end{bmatrix} \tag{5.25}$$

5.6 Remarks

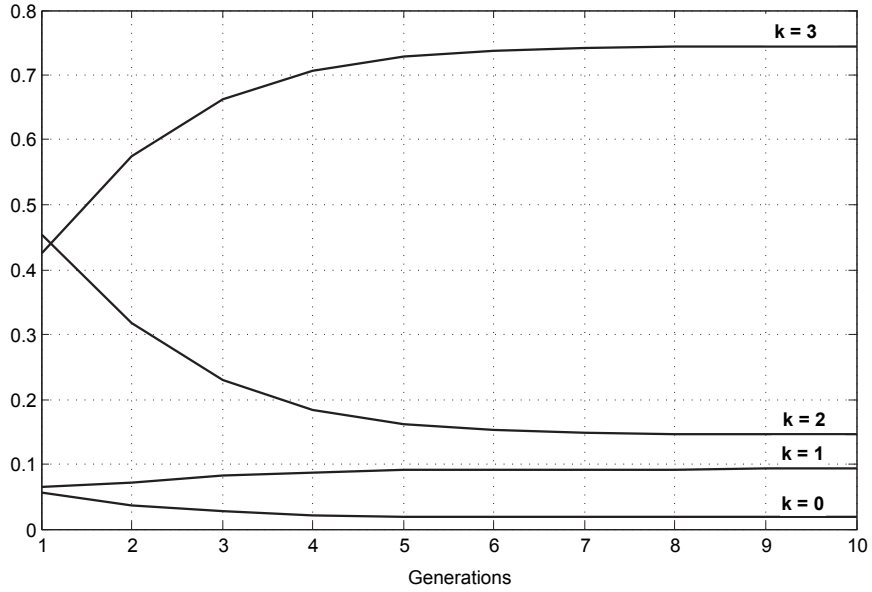


Figure 5.2: Expected population for 10 generations.

Fig. 5.2 shows the expected next generation population for all four chromosomes of length $\ell = 2$ in our numerical example. Chromosome **3** = (11) has the highest probability to survive in the entire solution space with a probability of 0.7441 after 10 generations. Chromosomes **0,1** and **2** have probabilities of 0.0176, 0.0924, and 0.1459 respectively.

As we can see from the example, the dynamical system model takes into account the construction and destruction of strings (chromosomes) by considering all possible

$f_0 = 0.7262$	$f_8 = 0.7262$	$f_{16} = 0.7262$	$f_{24} = 1.0$
$f_1 = 0.5446$	$f_9 = 0.3631$	$f_{17} = 0.7262$	$f_{25} = 1.0$
$f_2 = 0.5446$	$f_{10} = 0.1815$	$f_{18} = 0.5446$	$f_{26} = 1.0$
$f_3 = 0.1815$	$f_{11} = 0.0$	$f_{19} = 0.3631$	$f_{27} = 1.0$
$f_4 = 0.7262$	$f_{12} = 0.7262$	$f_{20} = 0.7262$	$f_{28} = 1.0$
$f_5 = 0.5446$	$f_{13} = 0.7262$	$f_{21} = 0.5446$	$f_{29} = 1.0$
$f_6 = 0.8$	$f_{14} = 0.5446$	$f_{22} = 0.3631$	$f_{30} = 1.0$
$f_7 = 0.1815$	$f_{15} = 0.9631$	$f_{23} = 0.0908$	$f_{31} = 1.0$

Table 5.1: Normalized fitness values for a chromosome of length $\ell = 5$.

ways of performing crossover and mutation operators in a GA. In the most general case (e.g. one point crossover) every string in the search space Ω can potentially be constructed. This effect can be captured by a matrix to represent the effects of recombination, where the strings or vectors are indexed from 0 to $(2^\ell - 1)$. The entry $(i, j, k)^t h$ of this array gives the probability that vectors i and j will recombine to produce vector k. The effect of mutation is similar to represent, but since only one string is involved, it is rather easier.

5.7 Estimating FGA behavior

We now apply the dynamical system model to predict the behavior of FGA in a typical scenario. The dynamical system model will help us to calculate the cumulative effects of GA operators as a population evolves through generations. Using the same mobility model introduced in Section 4.2, Fig. 5.3 displays five nodes distributed within an 8×8 hexagonal grid. We assume that all nodes remain in the same position all times except node N_3. N_3 runs FGA in order to optimize its position and consequently, NAC. Gray out cells are not within the transmission range of any MANET node.

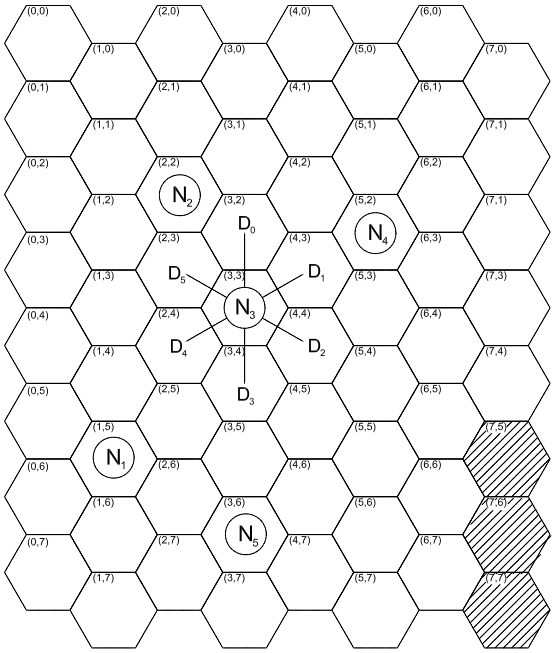


Figure 5.3: Example: mobile nodes at t with $R_{com} = 3$.

As mentioned in Section 4.3.1, each *chromosome* (candidate solution) is encoded using a bit string of length ℓ representing different combinations of the node's speed and direction. The chromosomes represent different combinations of node speed and direction. To keep the example simple, let us use chromosomes of short length than the 18-bit chromosomes adapted in our implementations, otherwise the calculations and matrices's dimensions will grown exponentially. In the hexagonal grid, N_3 can move in one of 6 different directions, therefore the node's direction and speed are encoded using 3 bits and 2 bits, respectively. Therefore, the chromosome length is set to $\ell = 5$, which requires a solution space Ω with a size of $2^{\ell} = 32$. Without using this simplification, the solution space of our model would be $2^{18} = 262,144$.

Tables 5.1 shows the normalized fitness values for chromosomes of length $\ell = 5$. For instance, chromosome $\mathbf{6} = (00110)$ represents the node heading Northeast with a

$$\mathcal{F}(p)_0 = \begin{bmatrix} 0.0614 \\ 0 \\ 0 \\ 0.3673 \\ 0 \\ 0 \\ 0 \\ 0.1836 \\ 0 \\ 0 \\ 0.1836 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0.204 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \quad \mathcal{C}(p)_0 = \begin{bmatrix} 0.0449 \\ 0.0091 \\ 0.0383 \\ 0.3105 \\ 0.0019 \\ 0.002 \\ 0.0118 \\ 0.19 \\ 0.0043 \\ 0.0005 \\ 0.1189 \\ 0.0454 \\ 0.0001 \\ 0.0001 \\ 0.0014 \\ 0.0108 \\ 0.0035 \\ 0.0003 \\ 0.0052 \\ 0.0225 \\ 0.0017 \\ 0.0016 \\ 0.0121 \\ 0.1546 \\ 0.0001 \\ 0 \\ 0.0057 \\ 0.0007 \\ 0 \\ 0 \\ 0.0002 \\ 0.0017 \end{bmatrix}, \quad \mathcal{U}(p)_0 = \begin{bmatrix} 0.0585 \\ 0.0041 \\ 0.0059 \\ 0.3511 \\ 0.0006 \\ 0.0018 \\ 0.0018 \\ 0.1801 \\ 0.0024 \\ 0.0001 \\ 0.1747 \\ 0.0053 \\ 0 \\ 0 \\ 0.0018 \\ 0.0018 \\ 0.0006 \\ 0.0001 \\ 0.0001 \\ 0.0055 \\ 0 \\ 0.002 \\ 0.002 \\ 0.1958 \\ 0 \\ 0 \\ 0.0018 \\ 0.0001 \\ 0 \\ 0 \\ 0 \\ 0.002 \end{bmatrix}$$

(a) (b) (c)

Figure 5.4: Expected population distribution after (a) $\mathcal{F}$, (b) $\mathcal{C}$, and (c) $\mathcal{U}$ operators for population vector p after one generation.

$$\mathcal{F}(p)_{40} = \begin{bmatrix} 0 \\ 0.0007 \\ 0.0011 \\ 0.0476 \\ 0 \\ 0 \\ 0 \\ 0.0024 \\ 0.0003 \\ 0.0156 \\ 0.0383 \\ 0.8919 \\ 0 \\ 0.0001 \\ 0.0003 \\ 0.0004 \\ 0 \\ 0 \\ 0 \\ 0.001 \\ 0 \\ 0 \\ 0 \\ 0.0002 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \quad \mathcal{C}(p)_{40} = \begin{bmatrix} 0 \\ 0.0014 \\ 0.0023 \\ 0.0539 \\ 0 \\ 0.0001 \\ 0.0001 \\ 0.0026 \\ 0.0009 \\ 0.0234 \\ 0.0447 \\ 0.8491 \\ 0 \\ 0.0003 \\ 0.0007 \\ 0.0093 \\ 0 \\ 0 \\ 0.0001 \\ 0.0014 \\ 0 \\ 0 \\ 0 \\ 0.0002 \\ 0 \\ 0.0002 \\ 0.0005 \\ 0.0087 \\ 0 \\ 0 \\ 0 \\ 0.0001 \end{bmatrix}, \quad \mathcal{U}(p)_{40} = \begin{bmatrix} 0 \\ 0.0013 \\ 0.002 \\ 0.0539 \\ 0 \\ 0.0001 \\ 0.0001 \\ 0.0029 \\ 0.0008 \\ 0.0235 \\ 0.045 \\ 0.8492 \\ 0 \\ 0.0003 \\ 0.0008 \\ 0.009 \\ 0 \\ 0 \\ 0.0001 \\ 0.0015 \\ 0 \\ 0 \\ 0 \\ 0.0002 \\ 0 \\ 0.0002 \\ 0.0005 \\ 0.0086 \\ 0 \\ 0 \\ 0 \\ 0.0001 \end{bmatrix}$$

(a) (b) (c)

Figure 5.5: Expected population distribution after (a) $\mathcal{F}$, (b) $\mathcal{C}$, and (c) $\mathcal{U}$ operators for population p after 40 generations.

speed of 2. As a result the node's next move could be toward cell $(5, 2)$. Notice in Table 5.1 that chromosome **6** with a high fitness value of $f_6 = 0.8$ among the other candidate solutions; since FGA favors smaller fitness values, chromosome **6** has little expectations that copies of itself will be included in forthcoming populations compared to, say, chromosome **10** with a fitness of 0.1815. Let us start with a random initial population P consisting of chromosomes **0**, **7**, **10**, **23**, and **30**, two copies of chromosome **3**, and three copies of chromosome **26**. Using Eq. (5.3), we can calculate $F(p)$ for FGA as shown in Fig. 5.4(a). After applying the selection operator to population P, chromosome **3**, with two copies in the initial population, has the highest probability (0.3673) of being selected for crossover. On the other hand, chromosome **26** with three copies in population P, has zero probability that copies of itself will be selected due to its high fitness value of $f_{26} = 1$. Recall that the goal of FGA is to find candidate solutions that minimize the fitness function shown in Eq. 4.2. Using Eq. (5.7) with a crossover probability of $c = 0.5$, we obtain $\mathcal{C}(p)$ as shown in Fig. 5.4(b). We observe from Fig. 5.4(b) that chromosome **3** still remains the individual with the highest probability to survive at the next generation after crossover operation. Also notice that most of the 32 chromosomes have non-zero probabilities compared with the values of $F(p)$ in Fig. 5.4(a), which implies that FGA developed a non-zero possibility of survival for many individuals after crossover operation. For a mutation probability $\mu = 0.01$, $\mathcal{U}(p)$ is calculated using Eq. 5.20 as shown in Fig. 5.4(c). Chromosome **3** has the highest probability of surviving and be a part of the next generation population, with probability of 0.3511.

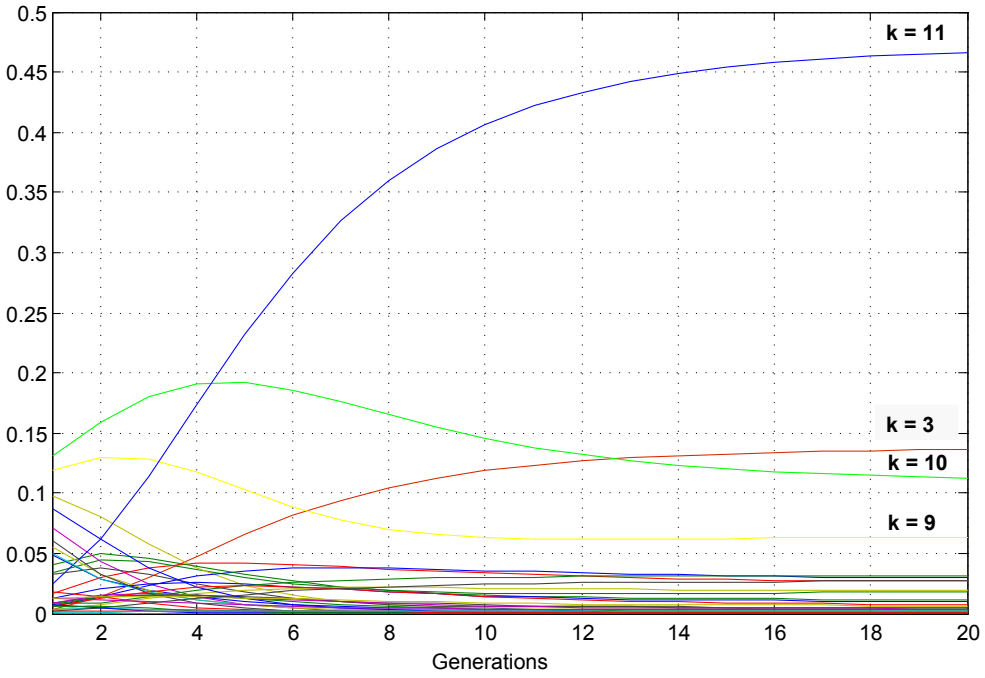


Figure 5.6: Expected population distribution for chromosomes of length $\ell = 5$ with $\mu = 0.01$ and $c = 0.5$, after 40 generations.

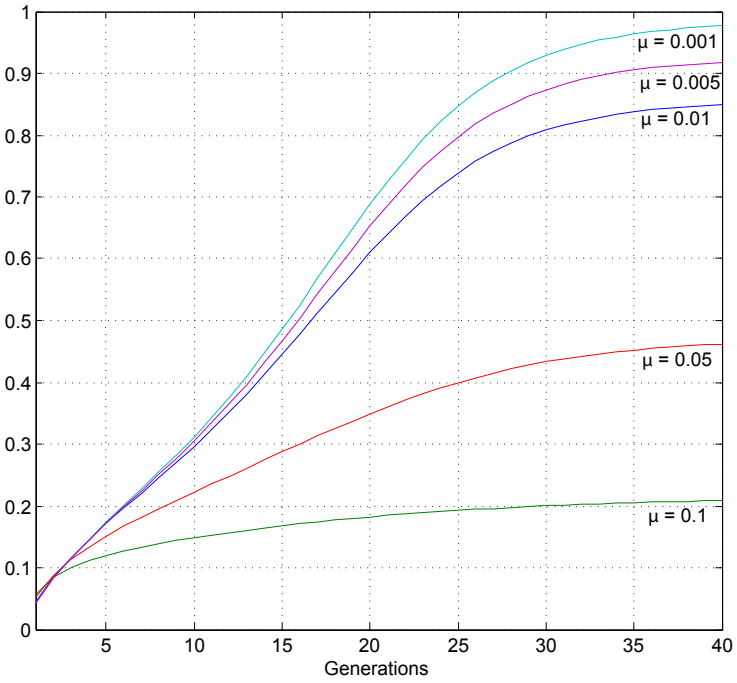


Figure 5.7: Expected population distribution of chromosome $\mathtt{11} = (01011)$ for different mutation rates μ after 40 generations.

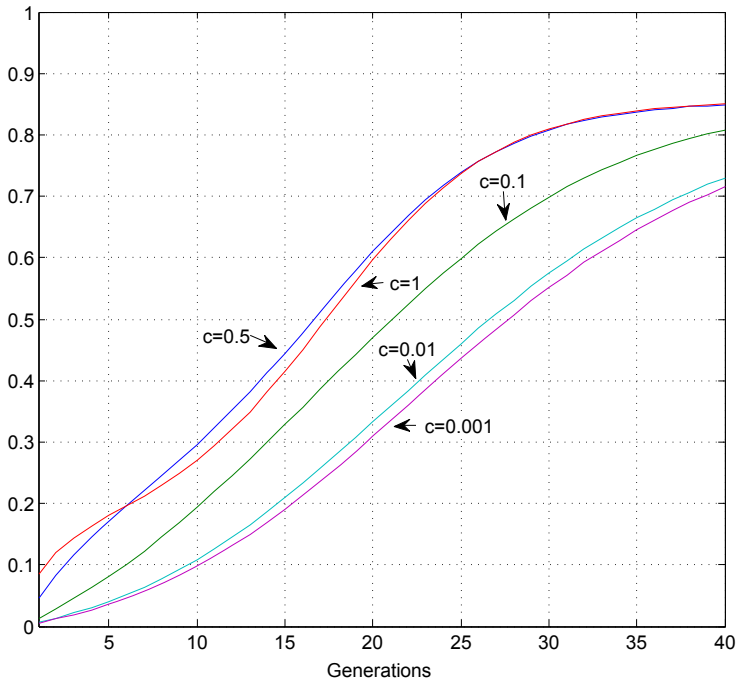


Figure 5.8: Expected population distribution of chromosome $\mathtt{11} = (01011)$ for different crossover rates c after 40 generations.

Fig. 5.5 shows the expected population distribution after 40 generations. Chromosome $\mathbf{11} = (01011)$ has the highest probability to survive in the entire solution space with a probability of 0.8491 after 40 generations. We observe the similar result from Fig. 5.6 where the expected generation population for all 32 chromosomes of length $\ell = 5$ during 40 generations are displayed. The probabilities of crossover (c) and mutation (μ) are set to 0.5 and 0.01, respectively. In Figs. 5.5 and 5.6 chromosome **11** represents N_3 bearing Southeast with a speed of 3: we expect N_3 to be located in cell $(6, 5)$ at $(t + 1)$ after running FGA (Fig. 5.9).

Figs. 5.7 and 5.8 examine the effect of mutation and crossover in FGA. They show the expected population distributions of chromosome **11** = (01011), the fittest chromosome in our example, for different mutation and crossover rates. Small changes in

the mutation rate μ cause a significant impact in the GA's trajectory (Fig. 5.7). As mutation increases, chromosome **11** gets weaker, and it is more likely that it (and any offspring) will die out. On the other hand, Fig. 5.8 shows that changes in the value of crossover rate c do not have a major impact on the expected population distributions of chromosome **11**.

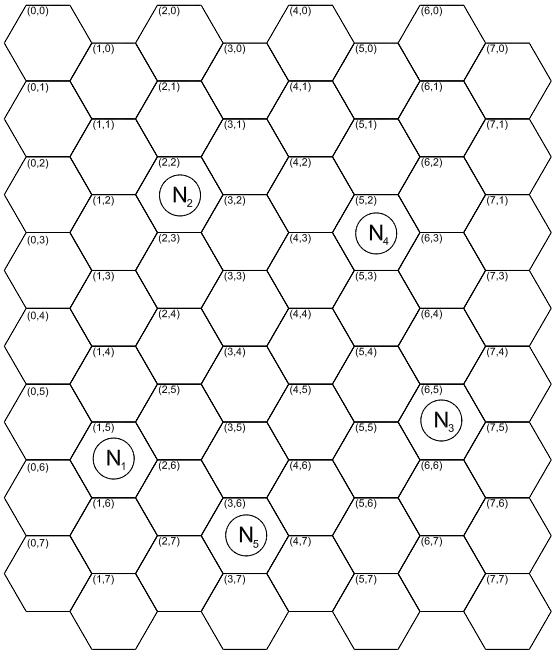


Figure 5.9: Expected N_3 location after running FGA at $(t+1)$ with $R_{com} = 3$.

In Figs. 4.1 and 5.9, white cells are covered by at least one mobile node while gray cells are not covered by any node. NAC improves from $61/64 \approx 95\%$ (Fig. 4.1) to $64/64 = 100\%$ after 40 generations of FGA (Fig. 5.9).

Chapter 6

Markov Chain Model for FGA

We show how to calculate the probabilities for all chromosomes (i.e., the candidate locations) of a mobile node for the next $(t+1)^{th}$ time unit by applying the dynamical system model to our FGA topology control algorithm in Chapter 5. The dynamical system model also gives a relatively safe indication of the FGA behavior since the trajectory of expected next population becomes less stochastic and more deterministic as the populations evolve through generations. Therefore, the heuristic function $\mathcal{G}(p)$ representing FGA introduced in Section 5.2 can be understood as a collection of random variables.

In this chapter, we introduce a Markov chain model to represent the different node configurations that a MANET with N nodes can occupy. In this model, a state s_i, where $s_i \in S = \{s_1, s_2, \ldots, s_r\}$, represents a geometric configuration of mobile nodes. It is assumed that multiple nodes cannot be in the same location and that if

at least one mobile node is isolated (i.e., the node has no neighbors) the geometric configurations are considered as invalid states. Fig. 6.1 shows examples of valid and invalid states for our Markov chain model when $R_{com} = 2$ and there are three mobiles nodes within the network. The states shown in Figs. 6.1(a) and 6.1(b) are considered valid states because the network is fully connected, meaning that all nodes in the network are reachable by other nodes through either one-hop or multi-hop communication. However, the states in Figs. 6.1(c) and 6.1(d) are considered invalid states since there is at least one isolated node in those configurations.

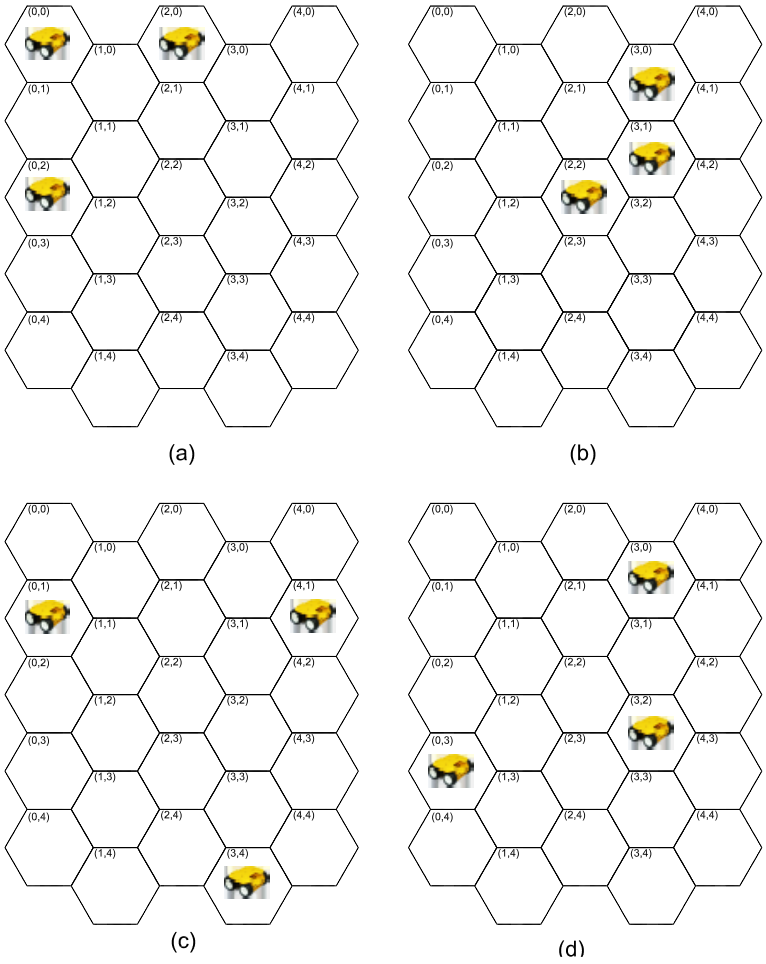


Figure 6.1: Examples of our Markov chain model for $R_{com} = 2$ and $N = 3$: (a,b) are valid states, while (c,d) are invalid states.

The number of valid states, S_{Val}, for a given MANET can be calculated as the number of permutations of a set without repetition minus the number of invalid states, S_{Inv}:

$$S_{Val} = \frac{(d_{height} \times d_{width})!}{N! \cdot (d_{height} \times d_{width} - N)!} - S_{Inv} \tag{6.1}$$

where $(d_{height} \times d_{width})$ is the size of the hexagonal grid and N is the total number of mobile nodes. Examples of the number of valid states for various communication ranges of R_{com} and hexagonal grid sizes are given in Table 6.1.

Table 6.1: Number of valid states for different R_{com} and hexagonal grid sizes.

	Hexagonal Grid sizes						
R_{com}	3×3	4×4	5×5	6×6	7×7	8×8	9×9
1	32	82	154	248	364	502	662
2	81	358	857	1554	2449	3542	4833
3	84	540	1752	3856	6819	10574	15121
4	84	560	2227	5916	12033	20614	31539
5	84	560	2298	6932	15995	30362	50309

A *transition matrix* Q is a numerical representation of a Markov model specifying the probability that a system will transition from one state s_i to another state s_j in a single time unit. A *right stochastic matrix* has an equal number of rows and columns and each of the rows has a sum of 1, thus signifying that there are no *hidden* states [54]. Given the locations of a set of mobile nodes at time t, the probability that they will move from a given configuration (i.e., a *state*) to any other configuration at time $(t + 1)$ can be calculated using the dynamical system model. Thus, for a given MANET, a one-step memoryless transition matrix for FGA can be generated. It is important to note that we are not concerned with the identity of the node that has

moved from one location to another, merely, the change in the geometric configuration of the mobile nodes in a given transition.

6.1 Homogeneous Finite Markov Chains

A homogeneous Markov chain on a finite space S having an initial distribution $\nu(s)$ (i.e., the probability of being in a particular state) has a transition matrix that is the same at every instant of time (i.e., $Q_i = Q$, where $i = 1, 2, \cdots$). Given this definition, the system is considered *memoryless*. The distribution of states $s \in S$ at times $t \geq 0$ is given by $Q^{(t)}(s_0, \cdots, s_n) = \nu(s_0)Q_1(s_0, s_1) \cdots Q_t(s_{n-1}, s_n)$. A useful subset of Markov chains includes those that are *ergodic* because it has been proved that they will eventually converge to a stationary distribution [54–56]. For a transition matrix to be ergodic, it must be both *irreducible* and *aperiodic*.

A Marvok chain is considered irreducible if all states can be reached from all others. In a Markov chain $(X_0, X_1, \ldots)$ with state space $S = s_1, \ldots, s_k$ and transition matrix Q, a state s_i communicates with another state s_j , writing $s_i \rightarrow s_j$, if the chain has positive probability of ever moving to s_j starting from s_i. If $s_i \rightarrow s_j$ and $s_j \rightarrow s_i$, we say that the states s_i and s_j intercommunicate, and write $s_i \leftrightarrow s_j$.

Definition 1: *A Markov chain $(X_0, X_1, \ldots)$ with state space $S = \{s_1, \ldots, s_k\}$ and transition matrix Q is said to be **irreducible** if for all $s_i, s_j \in S$ we have that $s_i \leftrightarrow s_j$; otherwise the chain is said to be **reducible**.*

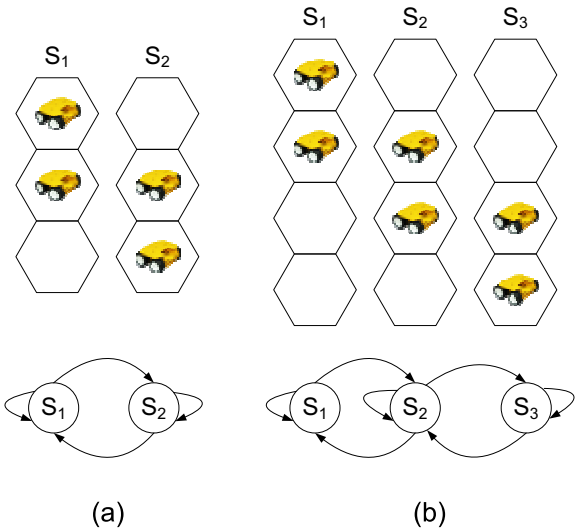


Figure 6.2: Markov chain models when $N = 2$ $R_{com} = 1$ for (a) 3×1 and (b) 4×1 hexagonal grids.

Definition 2: *A state s_i of a Markov chain is called* ***absorbing*** *if it is impossible to leave it (i.e., $Q_{s_i s_i} = 1$); a Markov chain is absorbing if it has at least one absorbing state, and if from every state it is possible to go to an absorbing state (not necessarily in one step).*

For instance, there are no absorbing states in any of the two Markov chains shown in Figs. 6.2(a) and (b) for our FGA topology control algorithm with different network parameters, since $Q_{s_i s_i} \neq 1$ for all $s_i \in S$.

Lemma 5: *A Markov chain model for our FGA topology control algorithm is irreducible if and only if $Q(T_{s_j} < \infty | s_0 = s_i) > 0$ for all $s_i, s_j \in S$ assuming $Q^0(s_0 = s_i)$ and T_{s_j} is the shortest number of steps from s_i to s_j (i.e., the probability of moving from one configuration state to another in finite time is non-zero).*

Proof: Suppose there exists integers n, m such that $Q^n_{s_i s_k} > 0$ and $Q^m_{s_k s_j} > 0$.

If $l = n + m$, then $Q^l_{s_i s_j} > Q^n_{s_i s_k} Q^m_{s_k s_j} > 0$ by using the Chapman-Kolmogorov equations [57]. Therefore, we conclude that the Markov chain can go from s_i to s_j by first going from s_i to s_k in n steps, and then (independent of the past) going from s_k to s_j in additional m steps. Moreover the state space S can be only partitioned in one communicating class C (that is, if every state is accessible from every other), hence $S = C$ and the chain (and its transition matrix Q) is said to be irreducible. ■

If we consider the chain in Fig. 6.2(b) for $N = 2$, $R_{com} = 1$ and a 4×1 hexagonal grid, wee see that the state space $S = \{1, 2, 3\}$ cannot be broken into disjoint subsets. Hence, there is only one communicating class $C = S$ where the set of states $C = \{1, 2, 3\}$ all communicate with one another.

For a finite or infinite set $\{a_1, a_2, \ldots\}$ of positive integers, $gcd\{a_1, a_2, \ldots\}$ is the greatest common divisor of $a_1, a_2, \ldots$. The **period** $d(s_i)$ of a state $s_i \in S$ is defined as $d(s_i) = gcd\{n \geq 1 : (Q^n)_{i,i} > 0\}$. In other words, the period of s_i is the greatest common divisor of the set of times that the chain can return to s_i, given that we start with $s_0 = s_i$. If $d(s_i) = 1$, then the state s_i is **aperiodic**.

Definition* 3:** *A Markov chain is said to be* ***aperiodic *if all its states are aperiodic. Otherwise the chain is said to be* ***periodic****.*

***Lemma* 6:** *In a Markov chain for our FGA topology control algorithm, $d(s_i) = 1$ for all states $s_i \in S$, hence the Markov chain is aperiodic.*

Proof: Any state is accessible from itself in the Markov chain for our FGA topology

control algorithm since 1-step transition probability $Q_{s_i s_i} > 0$ for each state $s_i \in S$, therefore the chain is aperiodic (See Fig. 6.2). ■

Lemma 7: *The analytical configuration-based Markov chain representation of our FGA topology control algorithm is both irreducible and aperiodic, hence ergodic.*

Proof: Lemmas 5 and 6 define ergodicity [54]. ■

6.2 Convergent Nature of Ergodic Homogeneous Finite Markov Chains

For a finite set S with distributions μ and ν on S, the *total variation* is defined as $\|\mu - \nu\| = \sum_n |\mu(s_i) - \nu(s_i)|$. For example, assume we are comparing a *fair* coin with an *unfair* coin. If $Pr_{fair} = [Pr_{heads} = 0.5, Pr_{tails} = 0.5]$ and $Pr_{unfair} = [Pr_{heads} = 0.75, Pr_{tails} = 0.25]$ then $\|Pr_{fair} - Pr_{unfair}\| = |(0.5 - 0.75)| + |(0.5 - 0.25)| = 0.5$. Dobrushin's contraction coefficient [58] extends this measure and provides a rough measure of orthogonality between the distributions in a Markov kernel (i.e., transition matrix). It is defined as

$$c(Q) = \frac{1}{2} \cdot \underbrace{max}_{s_i, s_j} |Q(s_i, \cdot) - Q(s_j, \cdot)| \tag{6.2}$$

where c is the contraction coefficient and Q is a transition matrix. This represents half the largest total variation between any two rows in the transition matrix. When

$c(Q) = 1$, any two rows (i.e. distributions) of the transition matrix are disjoint. When $c(Q) = 0$, all of the rows in the transition matrix $Q(s_i, \cdot)$ are equal. The application of these measures lead to the following profound statements (proofs are in Winkler [54]:

***Lemma* 8:** *(taken from [54]) Let Q_1 and Q_2 be transition matrices and let μ and ν be probability distributions: $|\mu Q_1 - \nu Q_1| \leq c(Q_1)\,|\mu - \nu|$, $c(Q_1 Q_2) \leq c(Q_1)c(Q_2) \Rightarrow$ $|\mu Q_1 - \nu Q_1| \leq |\mu - \nu|$, $|\mu Q_1 - \nu Q_1| \leq 2 \cdot c(Q_1)$.*

***Lemma* 9:** *(taken from [54]) For each transition matrix Q the sequence $(c(Q^t))t \geq 0$ decreases.*

***Lemma* 10:** *(taken from [54]) If Q is ergodic then the sequence decreases to* 0.

***Theorem* 3:** *(taken from [54]) For an ergodic homogeneous transition matrix Q on a finite space with a stationary distribution μ, then uniformly for all distributions ν, $\nu Q^t \to \mu$ as $t \to \infty$.*

Winkler demonstrates in Lemma 8 that the interaction of various distributions within a set with an ergodic system (transition matrix) reduces the orthogonality (i.e. the total variation) between them (assuming they are not disjoint). Lemmas 9 and 10 state that as a distribution iterates through an ergodic system it converges to a stationary distribution. Theorem 3 generalizes this result to include any initial distribution of the set. Using Theorem 3 we can state that the analytical model of our FGA topology control algorithm will converge to a stationary behavior:

***Theorem* 4:** *If the transition matrix Q for a Markov chain of our FGA topology control algorithm is ergodic, then Q will converge to a stationary distribution.*

Proof: As stated in Lemma 7, the transition matrix Q for a Markov chain of our FGA topology control algorithm is ergodic. Therefore, using Theorem 3, Q will converge to a stationary distribution.

In the following sections, we analytically find the convergence measure of various system configurations to support the statement given by Theorem 4. We also study the fitness of the final stationary distributions.

6.3 Convergence of FGA Analytical Model

In the analysis of the Markov model of our FGA topology control algorithm the mobile nodes are not allowed to occupy the same location, nor allow themselves to become disconnected (since they only move as far as one adjacent location per time unit). In this model, therefore, the likelihood that the set of mobile nodes will move from one configuration to another is only based on their current location. Furthermore, this transition is memoryless with regard to previous movements. For simplicity and without loss of generality, the number of mobile nodes within the network was set to three for each analysis presented here. The size of the hexagonal grid was varied between 3×3, 4×4 and 5×5 (any values larger have an exponentially larger number of states as shown in Table 6.1). The communication range R_{com} was varied as 1, 2

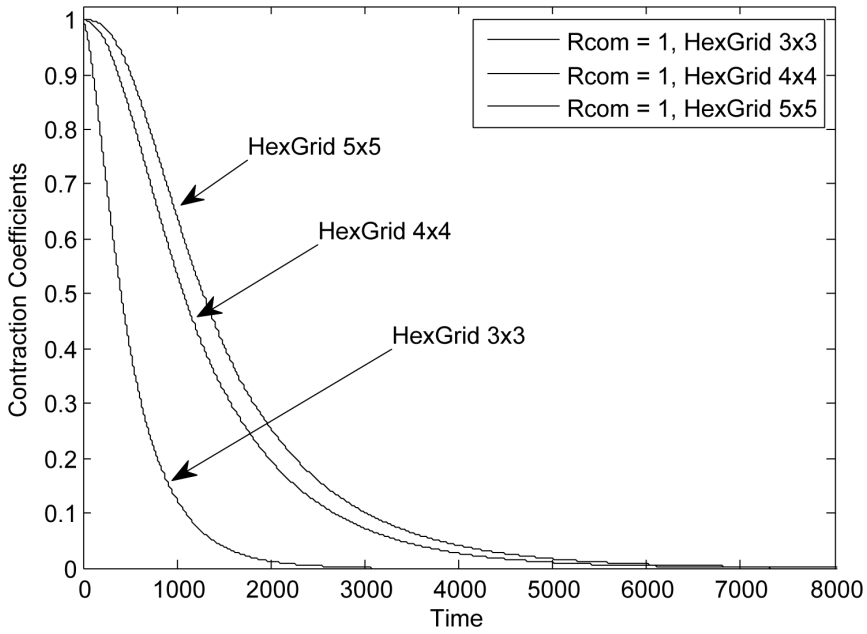


Figure 6.3: Contraction Coefficients for Analytical Model of FGA topology control algorithm when $R_{com} = 1$.

and 3 units.

Fig. 6.3 shows the Dobrushin's contraction coefficients iterated over time ($c(Q^t)$ as $t \to \infty$) for $R_{com} = 1$ and different hexagonal grid of sizes of 3×3, 4×4 and 5×5. As stated, the contraction coefficient gives a rough measure of the orthogonality (i.e., uniqueness) between the distributions (i.e., rows) in the transition matrix. For all parameter sets, the system the contraction coefficient goes to zero and the system converges towards a stationary distribution. In Fig. 6.3, the analytical analysis of our FGA topology control algorithm converge slower for larger hexagonal grids. This phenomenon can be better understood by looking at the two states shown in Fig. 6.4 where the probability of transition from the first state to the second in a single time unit is zero. Obviously, with each mobile node moving one space per time unit, it will take a minimum of five transitions to move from the first state to the second.

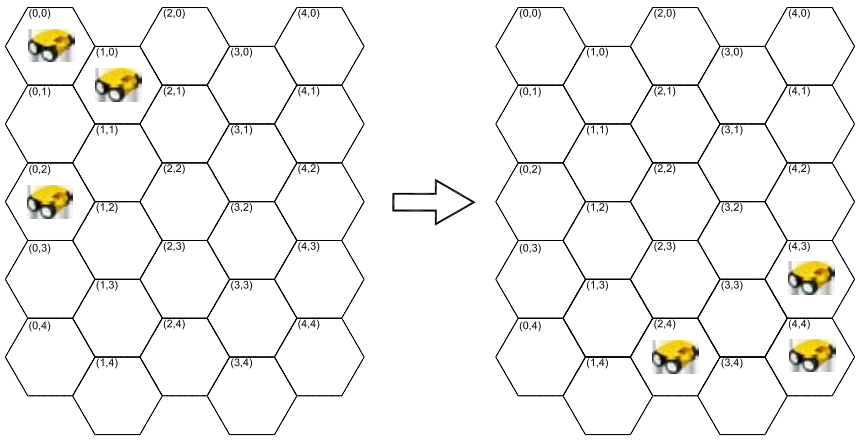


Figure 6.4: Example of unfeasible transition from one state to another in a single time unit.

Therefore, it will take several time steps for the iterated transition matrix Q^t to have a non-zero value for the displayed transition. Smaller hexagonal grids can accomplish this transition in fewer iterations and, therefore, converge to a stationary distribution faster.

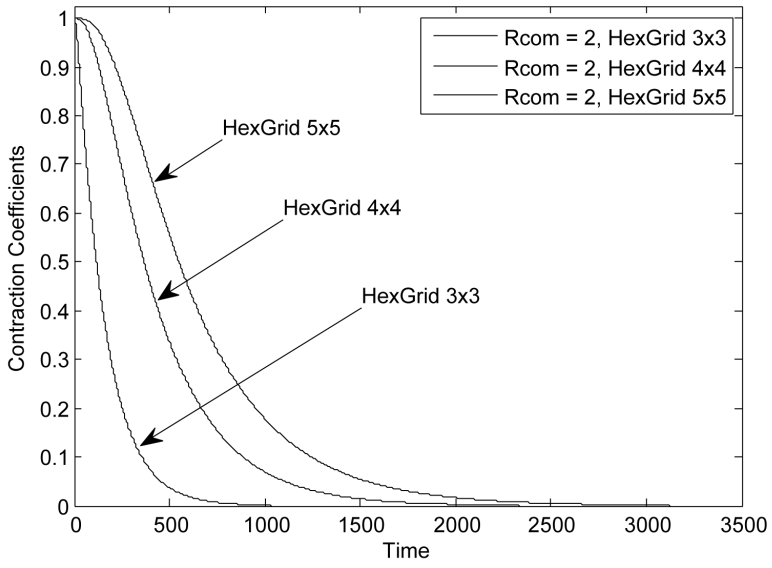


Figure 6.5: Contraction Coefficients for Analytical Model of FGA topology control algorithm for $R_{com} = 2$.

Figs. 6.5 and 6.6 demonstrate the same effect for $R_{com} = 2$ and $R_{com} = 3$ units,

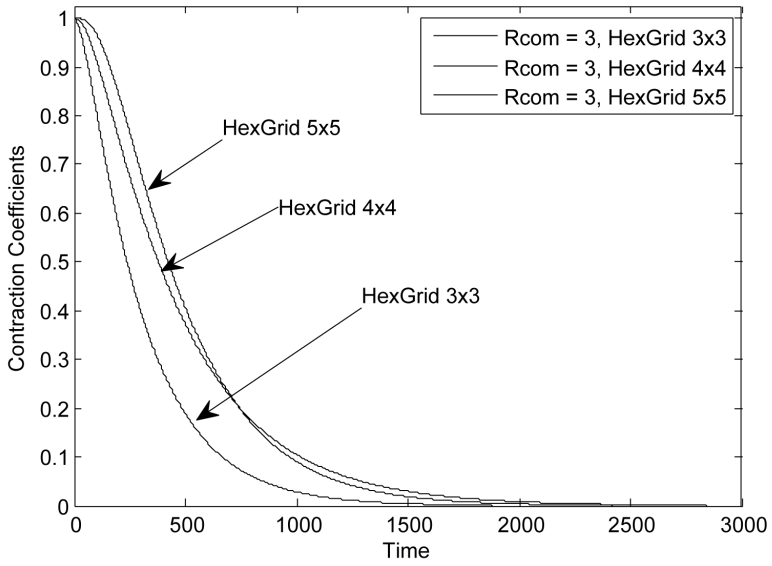


Figure 6.6: Contraction Coefficients for Analytical Model of FGA topology control algorithm for $R_{com} = 3$.

respectively. Many of the experiments converge in approximately 1,500 to 2,500 time units. It is important to realize that these convergence times are more dependent on the number of states (i.e. configurations) for a given parameter set than a direct correlation to the speed of convergence in an actual demonstration of the FGA. The importance of this measure is the demonstration that the system does in fact converge to a stationary distribution.

6.4 Fitness Analysis for Stationary Distribution

We demonstrated analytically in Section 6.3 that our FGA topology control algorithm converges to a stationary distribution. In this section we analyze the fitness of the final stationary distribution to determine if the FGA topology control algorithm will converge to a desired configuration. At first the total fitness for each state is deter-

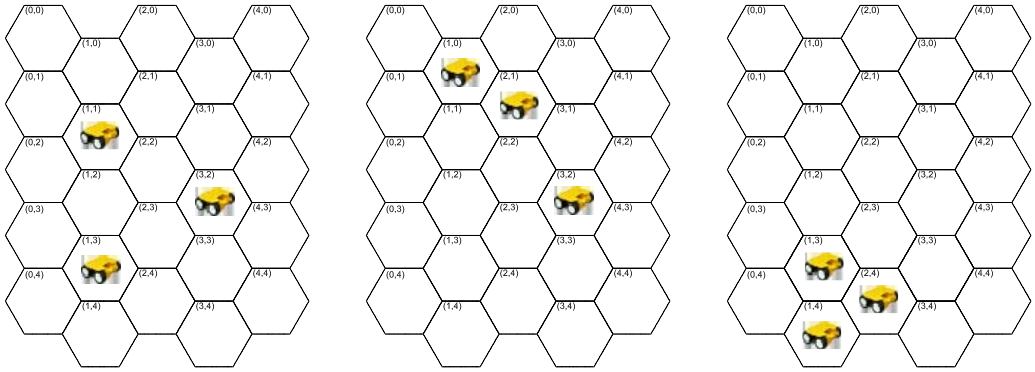


Figure 6.7: Example of (a) high, (b) medium and (c) low fitness state for $R_{com} = 2$, $N = 3$ and a 5×5 hexagonal grid.

mined by adding the individual fitness of each of the nodes in a given configuration. Since three nodes are used in each of the experiments, the total fitness value of 3 is denoted as a *high* fitness state. A total fitness value between 2 and 3 is denoted as a *medium* fitness state. If the total fitness is less than 2, it is referred to as a *low* fitness state. Using this notation, Fig. 6.7 shows examples of high, medium and low fitness states for $R_{com} = 2$, $N = 3$ in a 5×5 hexagonal grid.

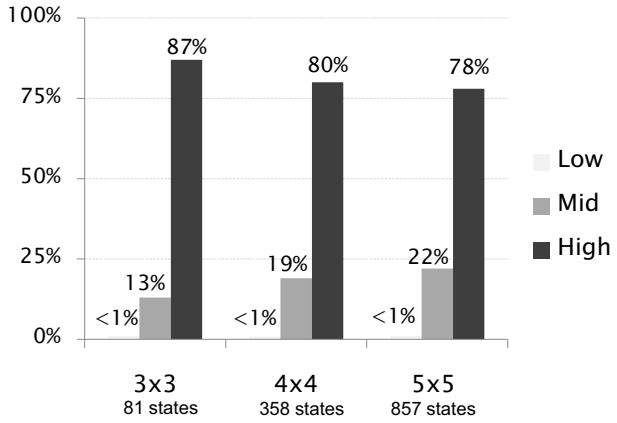


Figure 6.8: Aggregate total fitness of stationary distribution when $N = 3$ and $R_{com} = 2$ for 3×3, 4×4, and 5×5 hexagonal grids.

Fig. 6.8 shows the total fitness of the stationary distribution when $R_{com} = 2$, $N = 3$

and the hexagonal grid size varies as 3×3, 4×4 and 5×5. The first group of columns in Fig. 6.8 represents the aggregated fitness values for the system whose parameters are $R_{com} = 2$ and $N = 3$ in a 3×3 hexagonal grid. We can see that out of the 81 possible node configurations (i.e., *states*) the system can be in, there are less than 1% chance of being in any of the 48 low fitness states when the system converges to a stationary distribution. There is a 13% chance that the system will be in a medium fitness configuration (representing 48/81 states) and a 87% chance of being in one of the 5 high fitness states. This demonstrates that FGA topology control algorithm is in fact driving the system to a configuration that is desirable. For each of the three configurations in Fig. 6.8, the probability of the system stabilizing in a low fitness state is always less than 1%. The probability of being in a high fitness state when $R_{com} = 2$ for 3×3, 4×4 and 5×5 hexagonal grids is 87%, 80%, and 78%, respectively.

Given that our FGA topology control algorithm behaves as expected, it is understandable that there is a high probability of being in a high fitness state and a very low probability of being in a low fitness state. A significant percentage of states are in a medium fitness. For example, in Fig. 6.8, the probability of medium fitness states is between 14% to 22%. This group is composed of configurations that are either suboptimal configurations, or optimal one but are penalized for being close to the border. An example of an optimal-but-penalized state can be seen in Fig. 6.7(b). The probability of being in this state is 1.09%. Other optimal-but-penalized states create this meaningful probability for each of the medium fitness state distributions in Fig. 6.8. Low fitness states that have suboptimal configurations have a probability

in the order of $10^{-4}\%$.

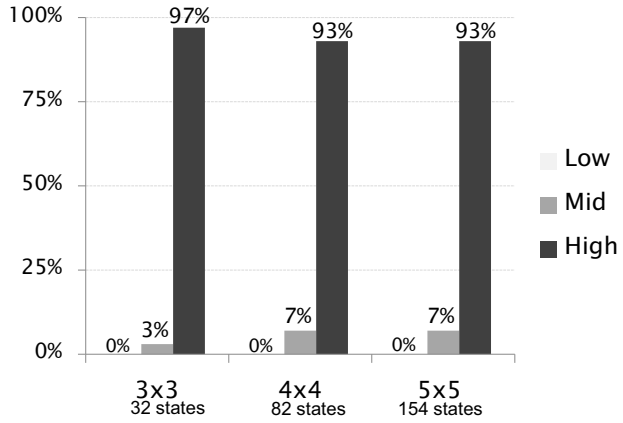


Figure 6.9: Aggregate total fitness of stationary distribution when $N = 3$ and $R_{com} = 1$ for 3×3, 4×4, and 5×5 hexagonal grids

The first three columns in Fig. 6.9 represents the aggregated fitness for the system whose parameters are $R_{com} = 1$ and $N = 3$ in a 3×3 hexagonal grid. As before, this chart shows that out of the 32 possible configurations the system can be in, when the system converges to a stationary distribution there will be a 97% chance that the system is in one of the 8 states with perfect fitness, a 3% chance that the system is in one of the remaining states with medium fitness and 0% chance that the system is state with low fitness (when $R_{com} = 1$ it is impossible to have low fitness without becoming disconnected). Similarly, the remaining two sets of columns in Fig. 6.9 demonstrate a probability of 93% high fitness, 7% medium fitness and 0% low fitness states.

Fig. 6.10 shows the total fitness of the stationary distribution when $R_{com} = 3$, $N = 3$ and the hexagonal grid varies from 3×3, 4×4 and 5×5, respectively. The columns in Fig. 6.10 representing 4×4 and 5×5 produced similar results as the $R_{com} = 2$

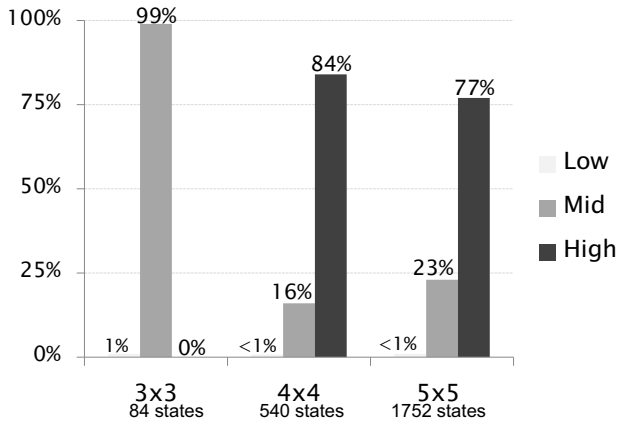


Figure 6.10: Aggregate total fitness of stationary distribution when $N = 3$ and $R_{com} = 3$ for 3×3, 4×4, and 5×5 hexagonal grids.

with the medium total fitness distributions slightly increasing. The columns for 3×3 in Fig. 6.10 are unique among the scenarios analyzed. This chart represents the final distribution for $R_{com} = 3$, $N = 3$, and a 3×3 hexagonal grid. It has a 99% of being in one state with a medium total fitness. This is due to the relatively large communication range compared to the small grid space. Essentially it demonstrates that when mobile nodes running our FGA topology control algorithm are over-crowded, they will attempt to find the best configuration in a suboptimal situation. The larger grid spaces for a similar communication range demonstrate that when crowding is not a problem, the nodes will most likely configure themselves in an optimal configuration.

Chapter 7

Simulation Experiments for Topology Control Algorithms

7.1 Simulation Software

We implemented a simulation software system in Java to study the effectiveness of FGA, our distributed topology control algorithms for uniformly distributing MANET nodes. Eclipse SDK version 3.2.0 was used as the development environment, and Mason, a fast discrete-event simulation library core developed in Java by George Mason University ECJLab, was used to model the mobile agents.

The simulation software implementation has approximately 5,000 lines of algorithmic Java code. To avoid possible inefficiencies, we developed our algorithms without using any existing GA libraries.

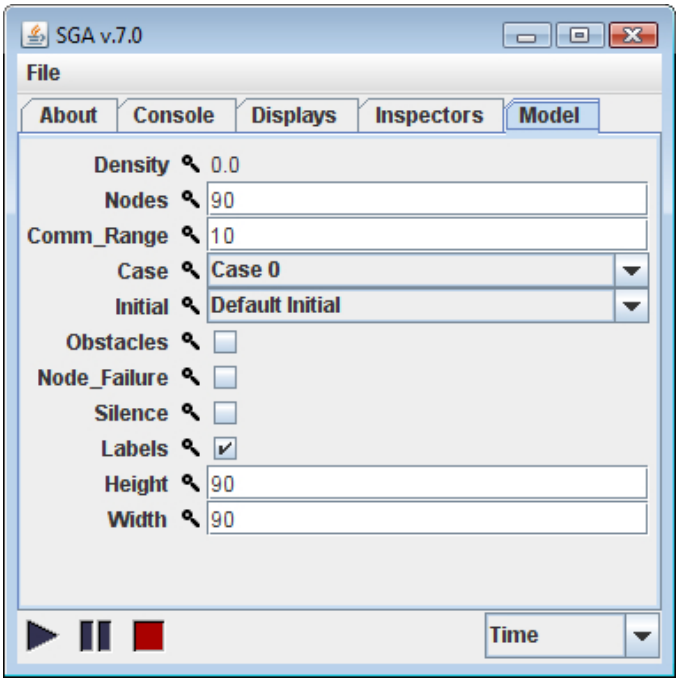


Figure 7.1: Graphical user interface for our GA software package: A screen shot of user input.

In our software, the behavior of mobile agent models can be customized by adding, removing, or changing features at any time. For example, a programmer can easily add different types of crossover or selection strategies to our model. Our simulation software is capable of running multi-agent applications which imitate a real-time topology control setup.

A sample screenshot of the graphical user interface of our simulation software are shown in Figs. 7.1 and 7.2. User-defined input parameters for our software include (Fig. 7.1):

- Nodes: the total number of mobile nodes,
- Comm_Range: the communication range of each mobile node (R_{com}),

- Case: type of evolutionary algorithms,
- T_{max}: the maximum number of iterations,
- Initial: initial deployment configurations. Currently there are three different initial deployment strategies for the mobile nodes: (*i*) start from the northwest corner, (*ii*) place the nodes randomly in a given area, and (*iii*) start from a given coordinate (e.g., the center of the area) in the terrain,
- Height: height of the geographical terrain,
- Width: width of the geographical terrain,
- obstacle inclusion (on user defined locations),

Fig. 7.2 shows a sample initial deployment of autonomous mobile nodes starting from the northwest sector of a given terrain. Note that a corner initial deployment option represents a more realistic approach of the topology control problem for the knowledge sharing mobile nodes than the other deployment possibilities over an unknown terrain. For example, in an earthquake rescue, a mine clearing mission, a military mission in hostile area, or a surveillance operation, all mobile nodes may be forced to enter the operation area from the same vicinity rather than random or central node deployment.

Our simulation software also has the ability to run experiments using a previously used initial mobile node distribution and initial conditions (i.e., the initial data for each mobile node includes a starting coordinate, speed, and direction). This ability is

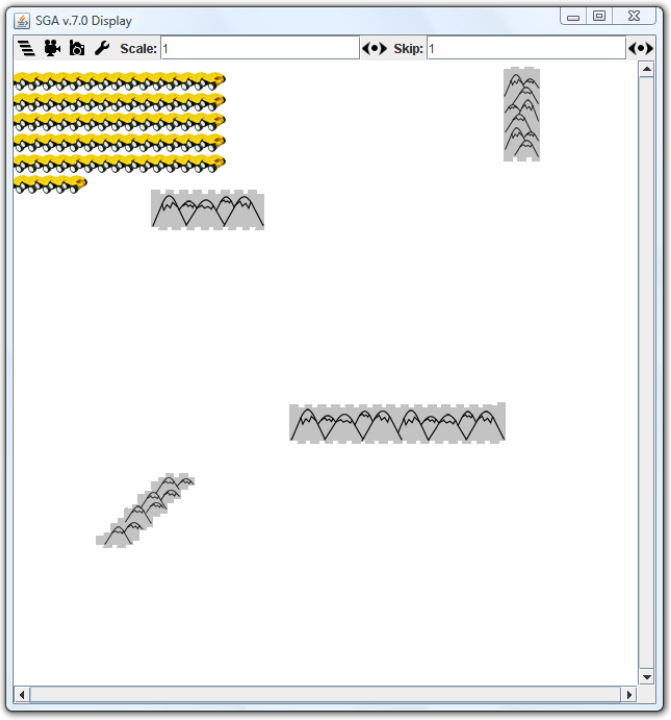


Figure 7.2: Graphical user interface for our GA software package: A screen shot from an initial mobile node distribution for FGA.

important since each experiment is repeated many times to eliminate the noise in the collected data and provide an accurate stochastic behavior of GA-based algorithms.

7.2 Simulation Experiments for FGA

For the simulation experiments, we consider 80 mobile agents with the same initial node distribution over an unknown region. Each mobile agent has a limited communication range (R_{com}), and, hence, can only be aware of its neighbors and the obstacles located in the node's sensing and communication range. The initial mobile agent deployment and the positions of the obstacles are shown in Fig. 7.2. We assume that it is not possible for the nodes to communicate through the obstacles.

We implemented FGA such that each mobile agent's movement is only affected by its current status of neighboring nodes. Due to this flexible implementation, we expect that each agent will be adaptive to the environment changes such as node failures, various terrain shapes and obstacles, and hostile attacks. To evaluate the performance and effectiveness of our FGA algorithm, we consider two types of applications [59]. In the first application, the mobile agents are deployed in a hostile region where some of the nodes are disabled during and after the deployment. In this application, the nodes are lost either due to equipment malfunctions (i.e., isolated and randomized losses) or hostile attacks (i.e., concentrated losses). The nodes affected by either malfunctions and hostile activity are considered to be disabled for the rest of that simulation experiment. After these losses, the remaining nodes must reconfigure their positions to compensate the missing area coverage due to lost team members.

In the second application, mobile agents intentionally stop communicating with the neighboring nodes located within R_{com} distance for short periods of time. This application, called the *silence mode*, simulates the conditions where the nodes need to go undetected by hostile forces. During the silent mode, the nodes cannot communicate with each other, and therefore, cannot modify their speed and directions. We assume that, during the silent mode, the nodes keep their direction and speed that they had before they entered the silent mode. Since their speed and directions remain uncorrected by FGA, we expect that NAC for will suffer during the silent mode. At the end of the silent mode, each node resumes communication and FGA becomes effective again.

7.3 Simulation Results for FGA

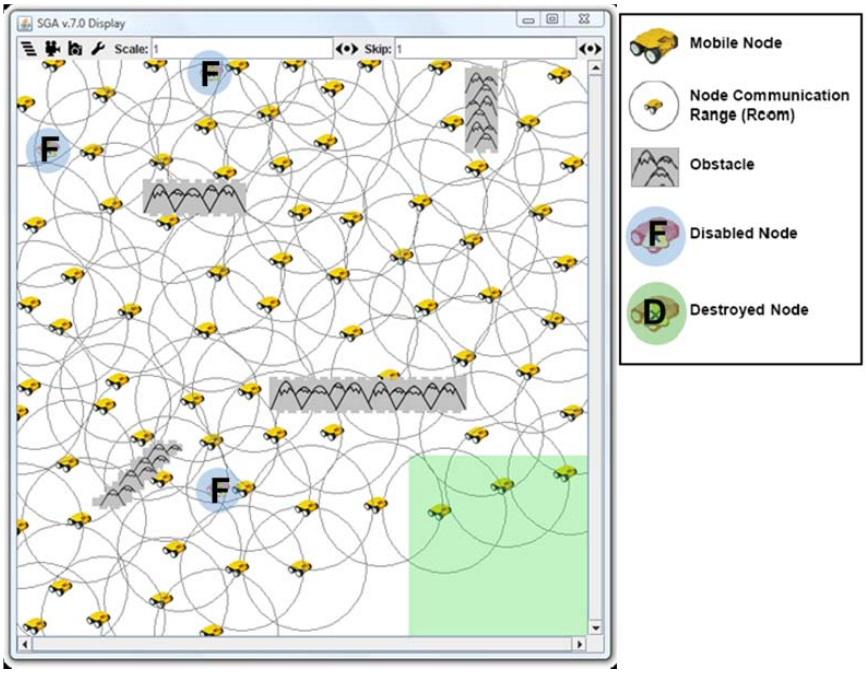


Figure 7.3: Mobile node distribution for Application 1 at $T = 400$ ($N = 77$ after three disabled nodes).

Fig. 7.3 shows the area coverage for a terrain which has arbitrary obstacles after 400 steps (i.e., iterations) of FGA. At this point of the experiment, there are three nodes that are disabled due to malfunctions (indicated by small solid circles with the letter F in Fig. 7.3). We can observe that, in spite of these obstacles, the mobile nodes using FGA obtain an almost uniform coverage of the area during the first 400 steps of Application 1.

At step $T = 401$, the first hostile attack takes place and destroys three mobile nodes, as shown in Fig. 7.4. The gray square at the south-east corner of the region represents the area where the enemy attacks take place. In addition, one more mobile agent experiences malfunction, reducing the total number of mobile nodes to $N = 73$.

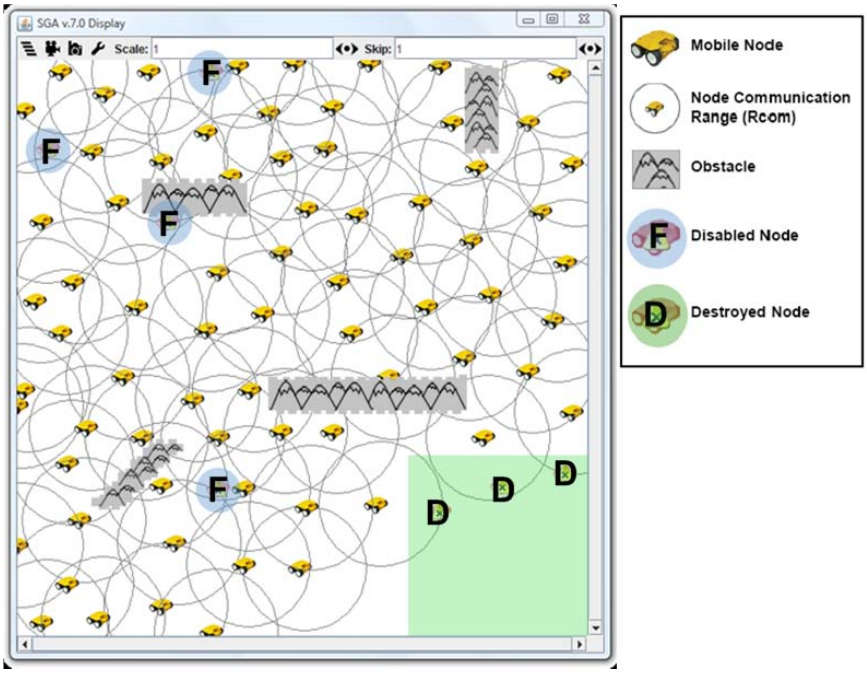


Figure 7.4: Mobile node distribution for Application 1 at $T = 401$ after the first enemy attack ($N = 73$ after four disabled and three destroyed nodes).

Fig. 7.5 shows the mobile node deployment at $T = 600$. Between $T = 401$ and $T = 600$, another mobile node becomes disabled due to malfunction. At this point, the number of remaining mobile nodes is $N = 73$ of which two nodes are in the hostile region.

At $T = 601$, the second enemy attack takes place destroying the two nodes in the hostile region while another node becomes disabled due to equipment malfunction, reducing the number of nodes in the experiment to $N = 69$. Fig. 7.6 shows the screen shot over the geographical area after the second enemy attack, which illustrates that the remaining mobile nodes keep performing FGA, and readjust their positions for a uniform area coverage.

The final mobile node distribution after running FGA for $T = 1000$ steps is presented

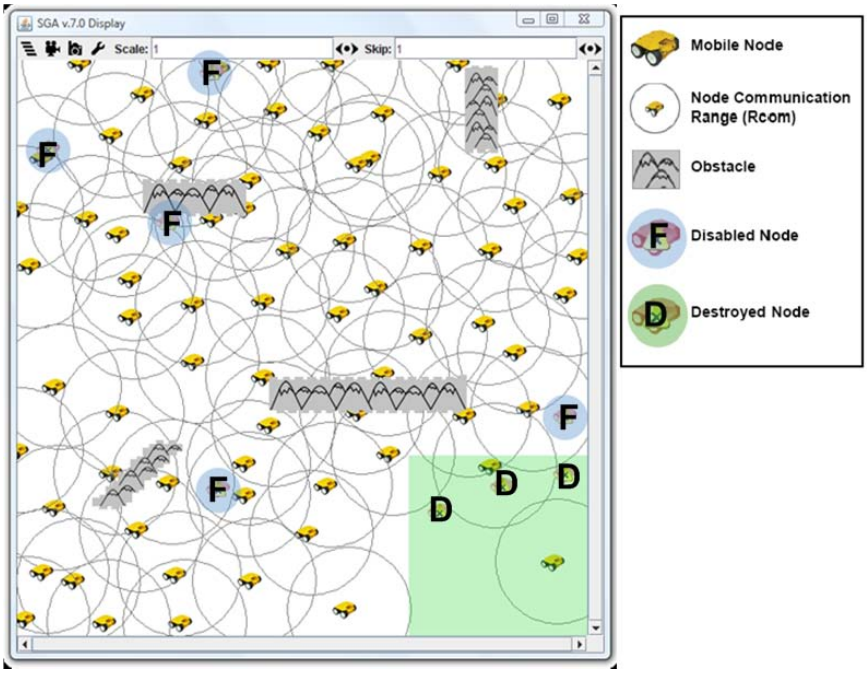


Figure 7.5: Mobile node distribution for Application 1 at $T = 600$ before the second enemy attack ($N = 72$ after five disabled and three destroyed nodes).

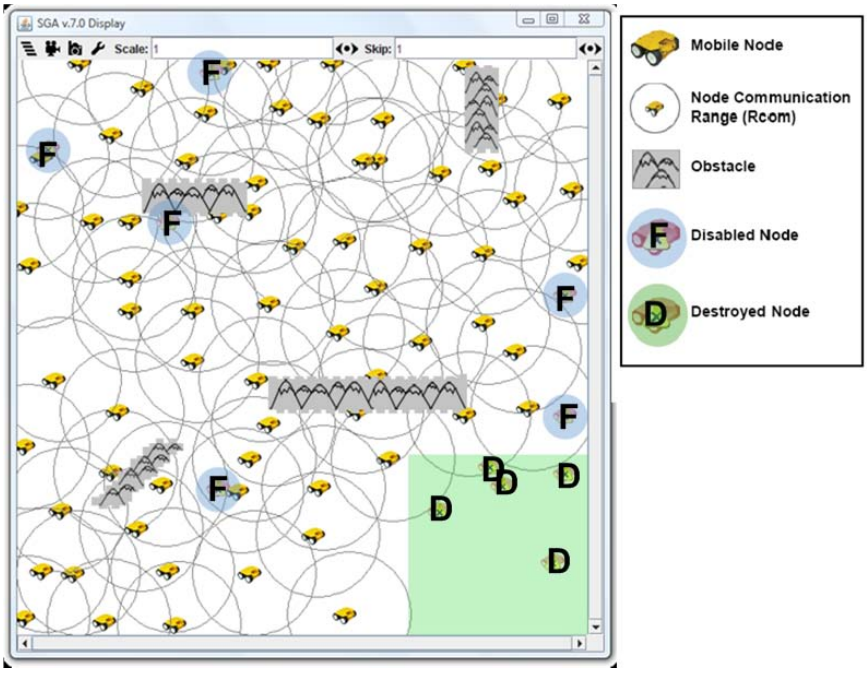


Figure 7.6: Mobile node distribution for Application 1 at $T = 601$ after the second enemy attack ($N = 69$ after six disabled and five destroyed nodes).

in Fig. 7.7, where the remaining nodes readjust their positions to compensate for the missing nodes. At this point two more nodes are disabled bringing the total of disabled nodes due to equipment malfunction to eight and total of destroyed nodes due to hostile attacks to five ($N = 67$). The network is considered fully connected at this point since all the nodes in the network are reachable by others through either one-hop or multi-hop communication.

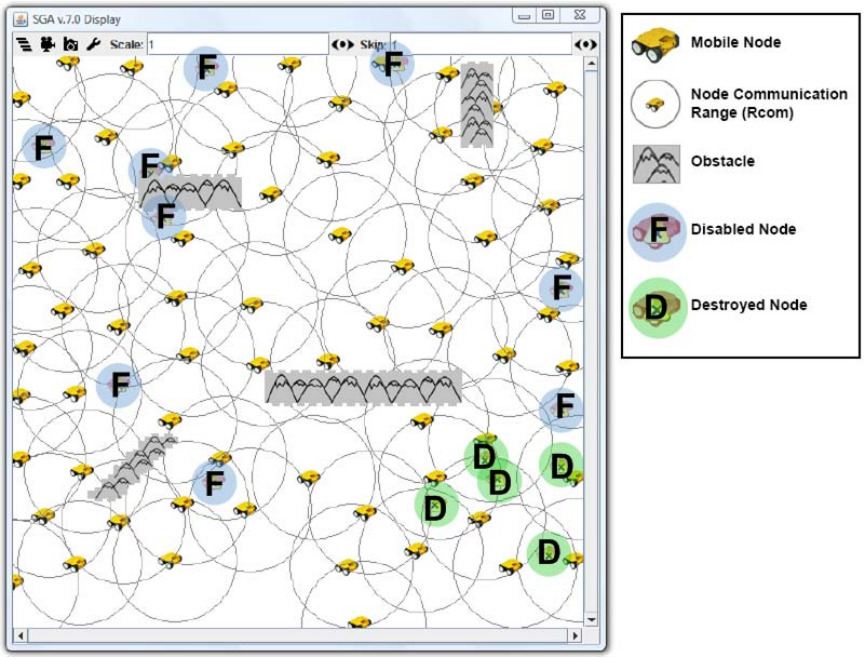


Figure 7.7: Final mobile node distribution at $T = 1000$ with $n = 69$ ($N = 67$ after eight disabled and five destroyed nodes).

Fig. 7.8 shows the convergence of FGA in terms of NAC through the iterations. The dark gray line in Fig. 7.8 illustrates that mobile nodes using FGA successfully deploy themselves around the obstacles if there were no hostile activity in the area, achieving a NAC value of 99% at $T = 1000$. Meanwhile, the light gray line represents the NAC when the nodes undergo malfunctions and hostile attacks. We can observe that the mobile nodes cover approximately 97% of the total area at $T = 400$.

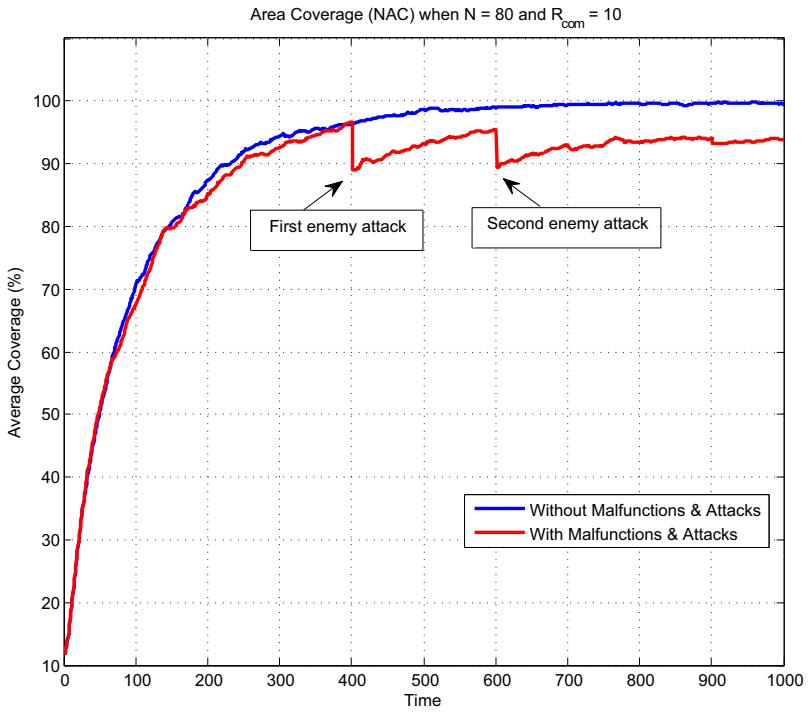


Figure 7.8: Convergence of FGA in terms of NAC after $T = 1000$ steps where $N = 80$ is dropped to $N = 67$ (at $T = 401$ and $T = 601$ two enemy attacks take place while a total of eight nodes become disabled due to equipment malfunction).

After the first attack at $T = 401$, there is a drop in the NAC value due to the lost seven nodes, which recovers after 200 steps ($T = 600$) reaching a NAC value of 95% (Fig. 7.8). Similarly, after the second attack, there is a drop in the NAC value at $T = 601$, which is then compensated by the remaining nodes after they reposition themselves using FGA approximately 300 steps after the second attack ($T = 1000$).

Fig. 7.9 shows the NAC for the *silence mode* application where the mobile agents intentionally stop communicating with their neighbors during short periods of time. In this experiment, the nodes perform FGA for 100 consecutive steps until $T = 100$ and then become silent for 200 iterations until $T = 300$. The silent mode is then repeated for 200 iterations between $T = 400 - 600$ and again in $T = 700 - 900$.

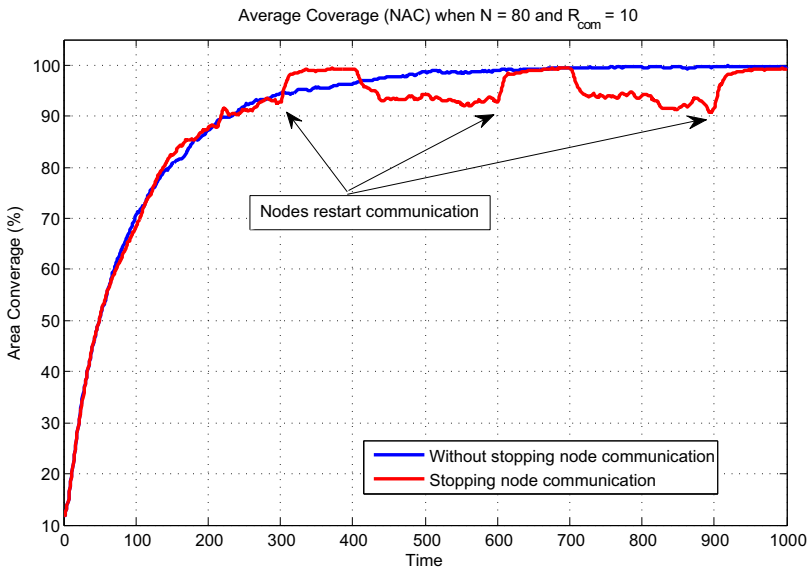


Figure 7.9: Convergence of FGA in terms of NAC after 1000 steps for Application 2 (the three silent modes are $T = 100 - 300$, $T = 400 - 600$, and $T = 700 - 900$).

During the silent mode, the nodes do not execute the FGA to correct their directions and speed, which results in reduced NAC values. We can observe that following silent periods, FGA significantly improves NAC values after $T = 300 - 400$, $T = 600 - 700$ and $T = 900 - 1000$.

7.4 Simulation Experiments for MDGA

In this section we describe the simulation experiment results for Mean-node Degree Topology Control Algorithm (MDGA) applications for different cases defined in Section 4.4.2. Fig. 7.16 shows the screen shot where the initial deployment of the agents start from the south-west sector. The mobile agents can have three different speeds (i.e., fast, slow, or immobile) towards any of the six possible directions. In these simulation experiments, the main input parameters include N, R_{com}, and $\overline{N}$. For all

MDGA cases defined in Section 4.4.2, a meaningful set of parameters are chosen to perform different experiments. To represent various network densities, we considered different numbers of mobile agents (i.e., N=80, 100, and 120) in a hexagonal region consisting of 10,000 cells ($n_{tot} = 50$). To observe the effects mobile agents' communication power, R_{com} values for each agent are varied as 8, 10, and 12. T_{max} for each experiment is selected as 1,000 steps so as to avoid possible transient results from the different GA algorithms. All experiments for each case from Section 4.4.2 are run for 20 times with different initial conditions, and the results are averaged to eliminate the random noise as much as possible. To provide a fair comparison, we use the same initial mobile agent deployment for all related cases. For example, the initial speed, direction and position of the mobile agents chosen in Case 1 experiment 1 is kept the same for the experiment 1 runs of Cases 2, 3, Random Walk (Case 0) and Hill Climbing. In total, our study includes 20 different initial conditions, each used for a different case. Fig. 7.16 shows one of the 20 different initial distributions of the mobile agents used in the experiments. As in the FGA experiments, this initial node distribution represents a more realistic deployment of the mobile agents into a given area.

7.4.1 Normalized Area Coverage

Fig. 7.10 the NAC as MDGA progresses in time. NAC is an important metric of convergence of MDGA towards a uniform node distribution over a given terrain. The average NAC for Case 2 is significantly better than the other cases (approximately

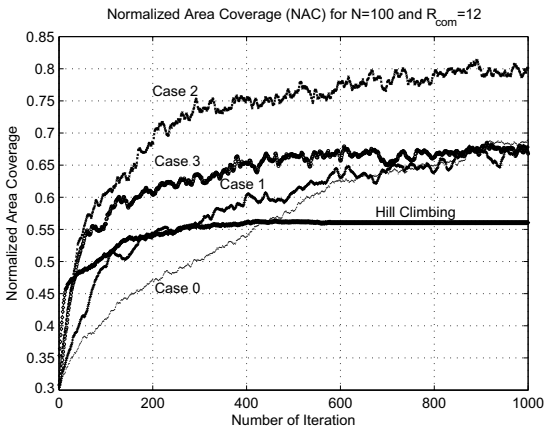


Figure 7.10: NAC vs iteration

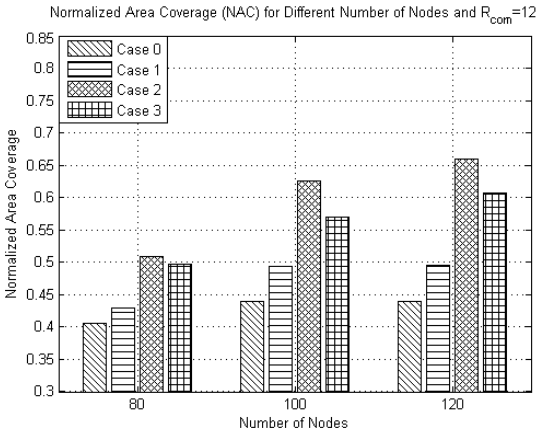


Figure 7.11: NAC for different N.

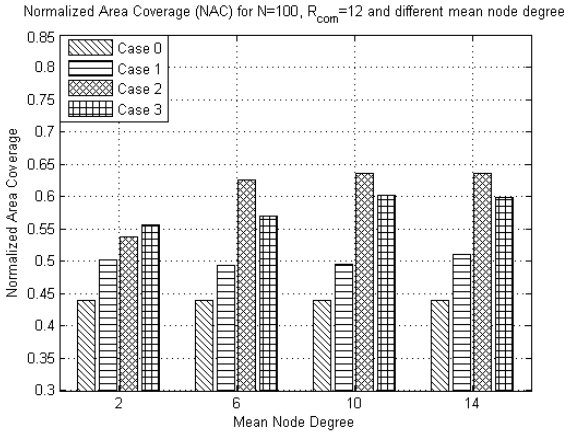


Figure 7.12: NAC for different $\overline{N}$.

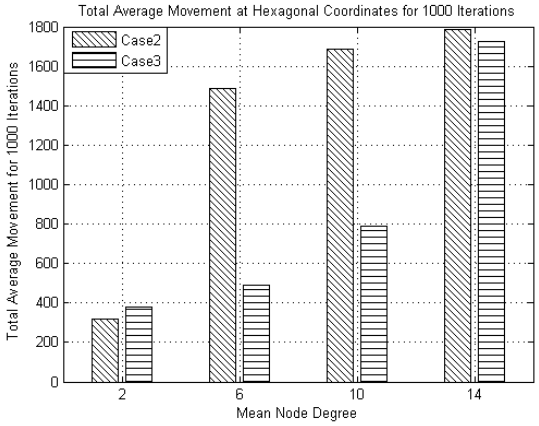


Figure 7.13: Average Movement for different $\overline{N}$.

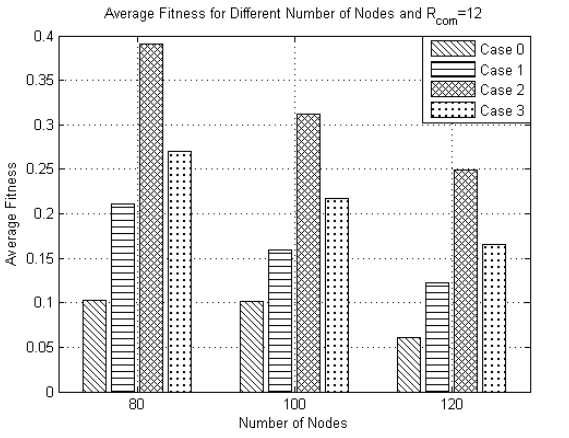


Figure 7.14: Average fitness for different N.

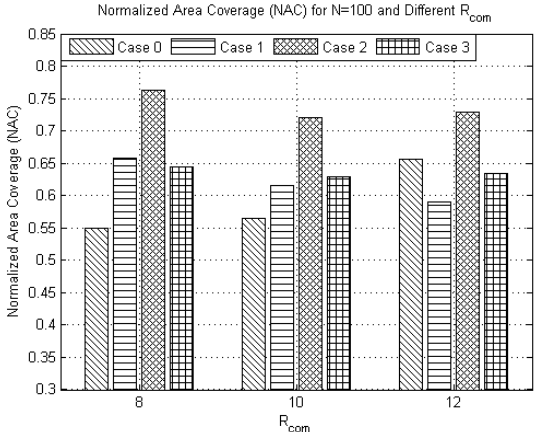


Figure 7.15: NAC for different R_{com}.

20%). Cases 1 and 3 converge towards similar average NAC values as the number of iterations grow since the mobile agents will have a better chance of spreading over the area. The convergence speed is the slowest for Case 0 (Random Walk) (approximately 900 iterations), since the mobile agents do not perform GA, and the fastest for Case 2 (approximately 300 iterations).

Clearly, all of the MDGA cases outperform Hill Climbing which has the lowest NAC value (Fig. 7.10). Based on the average of 20 experiment runs with Hill Climbing, we observe that it shows a promising start, however, the mobile agents running Hill Climbing fail to continue this improvement (i.e., cannot keep moving to positions with better goal function values), and instead become stuck on a local maximum (i.e., finding lower hills in the vicinity of their current locations). The average NAC for Hill Climbing reaches its highest point of 0.56 after $\approx$ 400 iterations, at which point all mobile agents stop and do not move again, whereas other cases continue to improve their NAC even after 800 iterations.

We also consider different number of mobile agents representing networks with different node densities. For $N = 80$, 100, and 120 with a fixed communication range of $R_{com} = 12$, the mobile agents calculate $\overline{N} = 5, 6$, and 7, respectively. We observe from Fig. 7.11 that the average NAC values proportionally increase as the number of nodes in the geographical area increases for all cases. This is an expected result because, as N increases, the nodes have a better chance of spreading uniformly over the area. The average NAC is the highest for Case 2, where the mobile agents stop moving when they reach the fitness of 1.0. A stopped node starts moving again when

its fitness value becomes less than 1.0 due to the nodes coming in or out of its vicinity. However, in Case 3, with the next higher average NAC value, the mobile node with fitness of 1.0 and its follower neighbors together stop moving and never move again. Hence, Case 3 achieves approximately 10% less NAC value than Case 2.

7.4.2 Effects of Mean Node Degree, Network Density and Communication Range

The significant impact of $\overline{N}$ value over MDGA performance is illustrated in Fig. 7.12 by using different values of $\overline{N}$. For $N = 100$ and $R_{com} = 12$, $\overline{N}$ is calculated as 6. We conducted experiments to obtain the average NAC values for $\overline{N}$=2, 6, 10, and 14, while keeping $N = 100$ and $R_{com} = 12$. Fig. 7.12 shows that the average NAC values do not improve even if $\overline{N}$ is incremented over 6. We also observe that the values less than the calculated $\overline{N}$ yield smaller NAC values.

Another important metric significantly dependent on $\overline{N}$ is the average distance traveled (and hence the power consumed) by each mobile agent. For $\overline{N}$=2, 6, 10, and 14, $N = 100$, and $R_{com} = 12$, we observe from Fig 7.13 that Case 3 is the most power conservative approach, where mobile agents visit less cells and spend less time and power to obtain the same NAC as compared to Case 2. As $\overline{N}$ is increased to 10 from the calculated value of 6, the traveled distances increase 11% for Case 2, and 60% for Case 3. The impact of $\overline{N}$=14 is more dramatic with 20% increase for Case 2, and 250% for Case 3 compared to the calculated value of $\overline{N}$=6.

Moreover, increasing the network density should have an inverse effect on the average fitness values since the fitness values of the mobile agents will be lower due to the overcrowded neighborhoods. Fig. 7.14 shows the average fitness values for $N = 80$, 100, and 120, using $R_{com} = 12$ for all cases. The average fitness value of each case proportionally decreases as the number of nodes in the region increases.

Using $N = 100$ and the calculated values of $\overline{N}$ for each case while changing R_{com} as 8, 10, and 12, we observed the effects of different communication ranges over the average NAC. In Fig. 7.15, the average NAC remains the same for different communication ranges since when analytical calculation of $\overline{N}$ includes the R_{com} as one of its parameters. Hence, as expected, the average NAC values remain unaffected for different cases.

7.4.3 Final Node Distributions

Sample final node distributions after applying all different MDGA cases for 1,000 iterations to the sample initial mobile agent node distribution given in Fig. 7.16 are shown in Figs. 7.17, 7.18, and 7.19 for Cases 1, 2, and 3, respectively. Each experiment is repeated 20 times for $N = 100$, $R_{com} = 12$ and the corresponding calculated value of $\overline{N} = 6$. Among the different cases, Case 2 obtains the best final node distribution. The second best result is obtained for Case 3, in which a leader stops itself and its followers when its fitness value reaches to 1.0. A possible disadvantage of Case 3 is that a leader may reposition its followers to new locations which are inadvertently

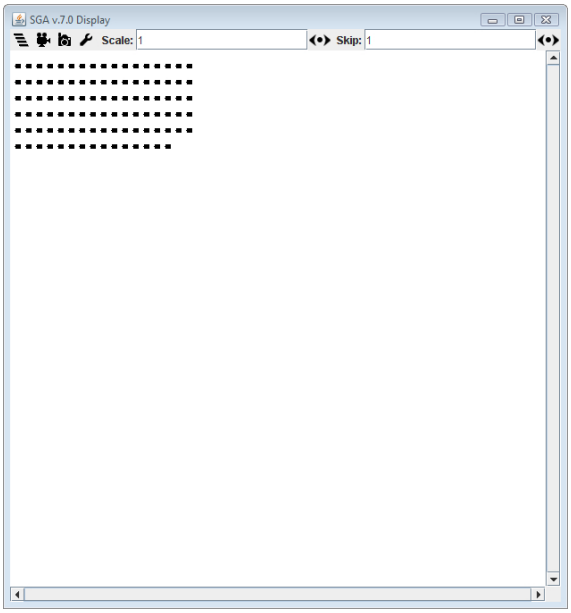


Figure 7.16: Initial distribution of mobile nodes.

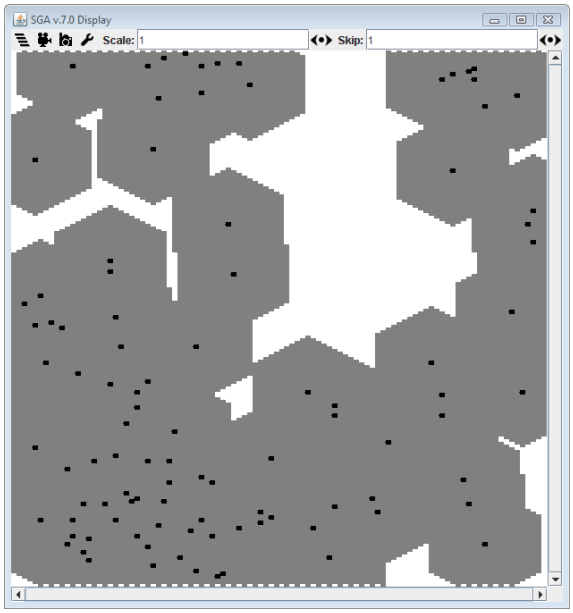


Figure 7.17: MDGA Case 1 after 1,000 steps.

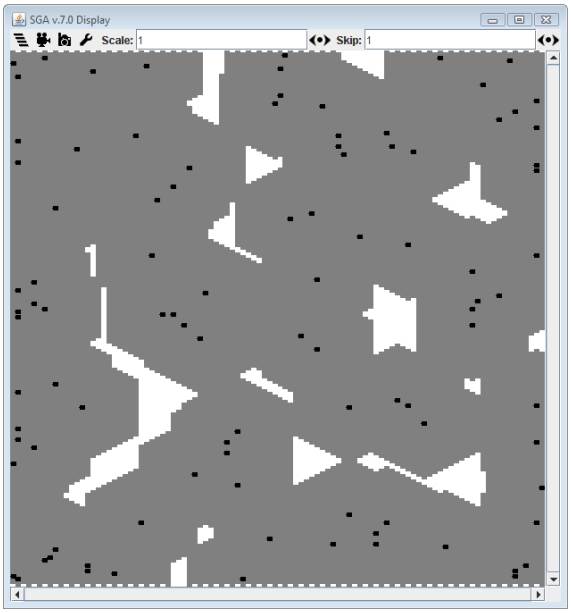


Figure 7.18: MDGA Case 2 after 1,000 steps.

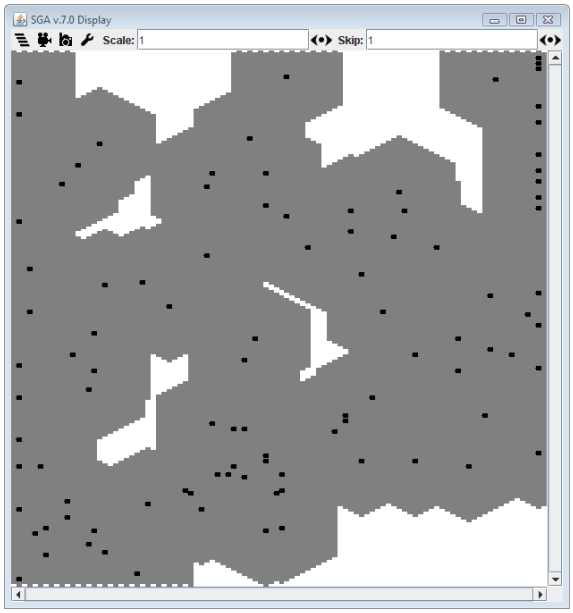


Figure 7.19: MDGA Case 3 after 1,000 steps.

close to another cluster. Using the simplest form of MDGA, Case 1 delivers the least uniform distribution among all the MDGA cases.

Chapter 8

Testbeds Implemented for Performance Analysis

Many research implementations of MANETs are developed as software tools that simulate network environments under very controlled conditions. Practical implementations of testbeds are often avoided because due to high cost of design and operation, and the difficulty of adapting real-time topology changes. Therefore, simulation is often used to determine the performance characteristics (such as scalability) of MANETs with large numbers of mobile nodes. Alternatively, having a realistic testbed can be used for proof of concept, determining physical characteristics (such as physical channel characteristics), determining unforseen implementation complications, and determining hardware and software requirements necessary to implement the MANET. Four testbeds were implemented to verify our GA-based topology control algorithms. They

include:

- Field-programmable gate array (FPGA) devices (*Virtex-II Pro* [2]),
- Small robotic units (*iRobots* [60] controlled by *gumstix* [61] processors with wireless capabilities),
- Virtual machine technology (VM*ware* [62]),
- Off-the-shelf laptops and PDAs.

8.1 FPGA Devices Testbed

To study the effectiveness of our GA-based topology control algorithms, we implemented a testbed with real wireless task oriented autonomous MANET nodes of FPGA devices [63]. The testbed consists of off-the-shelf wireless PCI cards that complies with the PCI Local Bus Specification version 2.3 (PCI v2.3) on the Xilinx ML310 with Power PC Virtex-II Pro Based Processor running VxWorks [64]. We choose wireless hardware support (Atheros AR521x chipset) under network devices 802.11a/b/g components and wireless mode support to include component support for ad-hoc IBSS to build the driver module into in VxWorks real-time kernel.

Both MANET communications and GA-based topology control algorithm are implemented as multi-threaded code and designed for multitasking operating systems that are optimized for real-time and embedded systems. It can run on Linux, Cygwin, and

VxWorks based real-time embedded platforms. This cross platform compatible implementation allows hybrid configuration and deployment of our testbed architecture such that communication between embedded mobile hand-held devices, navigators in vehicular ad-hoc networks, or multi-agents in general are possible and supported.

8.1.1 System Architecture

Virtex-II Pro hardware platform provides a powerful new paradigm for network processing with low latency. In many cases, Virtex-II Pro devices can offer higher overall performance than other solutions, including specialized network processors (NPs). The FPGA logic interrupts the PowerPC processor core only when processor instructions are needed for special packet types. By using the FPGA logic to process the most common packet types while the processor core handles the more specialized ones as a slave to the logic, the Virtex-II Pro architecture can provide higher overall performance than NPs, as well as more sophisticated processing capabilities than FPGA logic alone.

Fig.8.1 shows a high-level block diagram of the ML310 motherboard and its peripherals describing the hardware interface and components to the PowerPC 405 processor block. PowerPC 405 processor block is an embedded PowerPC 405D5 core with on-chip memory logic (OCM). The IBM PowerPC 405 core is integrated into the Virtex-II Pro device.

The ML310 Virtex-II Pro FPGA is connected to several peripherals either directly

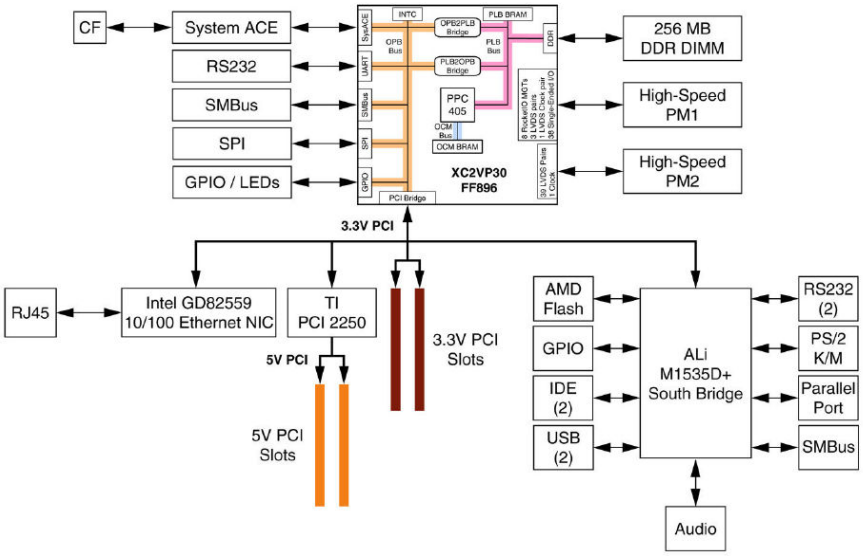


Figure 8.1: VirtexII Pro Based Xilinx ML310 High-Level Block Diagram [2].

or indirectly via the PCI bus. Board hardware consists of DDR DIMM memory, FPGA UART, System ACE CF controller, GPIO LEDs/LCD, SPI EEPROM, High-speed I/O through RocketIO MGTs, PCI bus interface which connects ALi M1535D+ PCI and Intel Ethernet/NIC PCI devices, and IIC/SMBus Interface for LTC1694 SMBus accelerator, RTC8566 Real Time Clock (RTC), 64 kb 24LC64 EEPROM, LM87 voltage/temp monitor, and DDR DIMM SPD EEPROM [2].

Compact flash card (CF) contains a System ACE file with an on-card intelligent controller that manages interface protocols, data storage and retrieval, ECC, defect handling and diagnostics, power management, and clock control. The ACE Flash card which appears as an additional hard drive also interfaces directly with the ACE Controller to provide a powerful pre-engineered configuration solution [65].

There are four components within the Virtex-II Pro Processor Block [66]:

- Embedded IBM PowerPC 405-D5 RISC CPU core
- On-Chip Memory (OCM) controllers and interfaces
- Clock/control interface logic
- CPU-FPGA Interfaces

The embedded PPC405 core is a 32-bit Harvard architecture processor with cache units, memory management unit, fetch decode unit, execution unit, timers, and debug logic unit [66]. The On-Chip Memory (OCM) controller serves as a dedicated interface between the FPGA block RAMs and the OCM signals contained within the embedded PowerPC 405 core [66]. The clock/control interface logic provides proper initialization and connections for PPC405 clock/power management, resets, PLB cycle control, and OCM interfaces [66]. The PPC405 core accesses high-speed system resources through Processor Local Bus (PLB) interfaces on the instruction and data cache controllers [66].

Our testbed architecture implements a typical MANET where each node in topology has autonomous mobility and wireless communication characteristics. All nodes run the same application and are configured with identical wireless characteristics. The nodal capabilities are assumed to be homogeneous. Mobility centric dynamic reconfiguration with respect to direction and speed is part of the MDGA topology control algorithm. Changes in the neighborhood information due to mobility are detected locally by each node through the heartbeat mechanism.

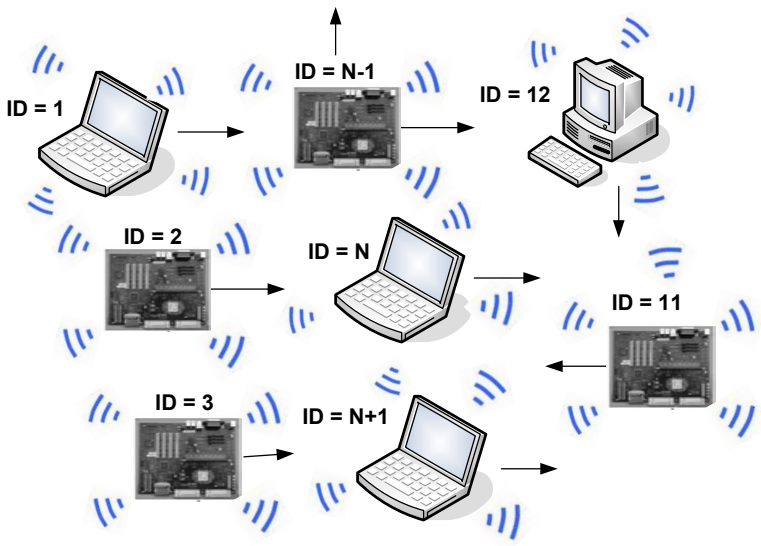


Figure 8.2: Testbed Network topology.

The testbed is portable for well defined VxWorks Board Support Packages [67], multi-threaded and designed for multitasking operating systems that are optimized for real-time and embedded microprocessor systems. Its ability to run on LINUX, Cygwin, and VxWorks based real-time embedded application platforms is verified (Fig. 8.1).

8.1.2 Network Topology

Our testbed environment consists of mobile nodes with homogeneous network communication properties all configured with the same ad-hoc group SSID located in an indoor laboratory environment. There are a total of 12 physical nodes with 3 Xilinx FPGA boards, 7 desktops, and 2 laptop computers as illustrated in Fig.8.2. The size of the geographic area is initialized to 500x500 units in our experiments. For simplicity, the geographic area is assumed to have no obstacles over a flat surface, where the node mobility is constrained by the configured boundaries.

8.1.3 Testbed Configuration

In our testbed, we used Xilinx FPGA family boards, in particular Power PC Virtex-II Pro Based Processor Xilinx ML310 running VxWorks, laptops and desktops running Cygwin under Windows. We used off-the-shelf wireless PCI cards that complies with the PCI Local Bus Specification version 2.3 (PCI v2.3) on the Xilinx ML310 board. We choose wireless hardware support (Atheros AR521x chipset) under network devices 802.11a/b/g components and wireless mode support to include component support for ad-hoc IBSS in VxWorks real-time kernel.

Our testbed configurations at present consists of low, normal and fast speeding nodes and maximum transmission ranges of 50, 75 and 100 units. Several of the configured default constants in our testbed implementation are related with the expiry times serving as timeout triggers. For example, timeout values are configured at initialization and currently set for 1, 2 and 3 seconds for tournament, crossover and mutating states respectively. *NITEXPIRYTIME* is a configurable attribute that indicates the longest time allowed since the last heartbeat message from a neighbor. For instance, if no heartbeat message received from a neighbor within the last NITEXPIRYTIME, that neighbor is purged from the neighborhood table. To cover cost/benefit ratio for mobile node's reachability and UDP packet drops (heartbeat message loss) due to wireless characteristics, NITEXPIRYTIME can be considered as a refresh cycle to adjust the granularity for which the neighborhood table gets updated. This provides configurable effective mechanism to find neighbors in the range with relative probability that multicast messages for tournament and crossover requests would be

responded regardless of positive or negative acknowledgments.

If a mobile node becomes isolated, the GA-based topology control algorithm is not active until a new neighbor is detected. Area coverage by a mobile node is determined by the circular transmission range and the configuration specific parameters defined for each scenario. All mobile nodes run identical configurations concurrently (but not synchronized) with some offset at initialization time.

8.1.4 Neighborhood Discovery

Dissemination of node coordinates along with node ID and the last update time parameter are part of the periodical heartbeat broadcasts by each node. In our experiments, heartbeat messages are broadcasted every 10 seconds.

The heartbeat broadcasts as well as tournament and crossover multicasts are ignored at the receiving end if the calculated distance between a source and destination is greater than the maximum communication range. Otherwise, the receiving node updates its neighborhood table with the heartbeat message information. Tournament and crossover multicast requests as well as acknowledgments for unicasts follow the same rule.

Angle	0	45	90	135	180	215	270	315
Encoding	000	001	010	011	100	101	110	111
Direction	E	NE	N	NW	W	SW	S	SE

Table 8.1: Binary encoding of configured directions used in testbed.

Speed	1	5	10	15	20	25	30	35
Encoding	000	001	010	011	100	101	110	111

Table 8.2: Binary encoding of configured speeds used in testbed.

8.1.5 Mobility Emulation

In order to emulate communication among mobile nodes, the TCP packets exchanged among the nodes are dropped intentionally when the distance between a source and a destination nodes is greater than a permitted communication range. Each mobile agent has capability to move to eight directions in three speeds within the geographic area (see Tables 8.1 and 8.2). The movement of a mobile node is independent and do not rely on any centralized controller or decision unit in the system. Initially, the nodes start in the middle of the geographical area and, for the first 150 seconds, move at random by changing direction every 10 seconds to give a chance to the nodes to spread. After 150 seconds, each node invokes its own GA-based topology control algorithm and broadcasts heartbeat messages every 10 seconds to exchange the genetic information required by the topology control algorithm to decide their speed and directions.

8.1.6 Testbed Experiments

We implemented the distributed *Mean-node Degree Topology Control* (MDGA) algorithm to be used within this testbed in a MANET environment at the City College of the City University of New York. We wrote a total of approximately 9K lines of C++ code along with approximately 800 lines of MATLAB code. We followed cross platform compatibility guidelines and have VxWorks for Xilinx Virtex II ML310 and Linux compatible code that currently operates on LINUX emulator on Windows. When a tournament multicast is issued, a request is sent to all of the nodes in the neighborhood table at that time. The first response with a positive acknowledgment determines the pairing node. Tournament operation is performed on each node independently. If a node is *less fit* than its partner, this node must adopt the speed and the direction of the *fitter* node. Then, a crossover multicast request is sent to the neighbors, where the first positive response determines the pairing node. Crossover operation, followed by the mutation are then performed on each node independently by each pair.

Since all the nodes have identical radio frequency ranges, there is no overlapping coverage between any two mobile nodes if the distance between them is greater than 2^*R_{com} units. Similarly, for distances less than R_{com}, any two nodes can potentially form a pair to successfully perform communication for tournament, and crossover operations.

We used speed and transmission range as the main parameters to evaluate the effec-

tiveness of MDGA for self spreading mobile nodes in MANETs. Currently, for low, normal and fast speeding nodes, and R_{com} values of 50, 75 and 100 units, there are a total of nine different settings. Each experiment is run for 1,500 seconds and repeated for ten times with the same initial values for the parameters of node speed, transmission range, and direction, and with the same initial node deployment positions to obtain accurate results and smooth out the noise in the output data. Each node gathers its own metrics every 10 seconds such as the node location, direction, speed, and fitness value.

As mentioned earlier, during an experiment, a mobile node can move into one of the eight different directions and with three different speeds. The number of nodes in the geographic area is 22 for all cases. We vary the transmission range to observe its effects on area coverage.

8.1.7 Effect of R_{com} on NAC

We used different R_{com} values in the experiments to observe the effects of MDGA in different network densities. For larger R_{com} values, more nodes can communicate with each other, emulating a dense network. Similarly, a small R_{com} implies that the network is sparse since fewer nodes can find a pair to perform MDGA operators. Figs. 8.3(a), (b) and (c) show NAC for R_{com} values of 50, 75 and 100 units, respectively. The maximum speed values of 5, 10 and 20 units/sec are used in the experiments. In all cases, NAC increases with the time, which demonstrates that MDGA is converging

toward a uniform node distribution.

8.1.8 Effect of Speed on NAC

Figs. 8.3(a), (b) and (c) indicate that the impact of speed on coverage is minor (compared to the effect of R_{com}) and that the increased node speed only results in faster convergence rate, but not necessarily in a better network coverage. However, the convergence speed is an important metric for battery consumption in MANETs. If a faster rate of convergence is needed for an application, the nodes should increase their speeds, consuming more battery power. However, if the final coverage value is the key goal, mobile nodes can lower their speeds to save battery usage since they will reach the same coverage a little later. This observation is also supported in Fig. 8.3(d) which shows the total cumulative distance moved by all nodes for $R_{com} = 50$.

Fig. 8.4 shows the locations of the mobile nodes in the area (in terms of their Cartesian coordinates and R_{com}) as snapshots taken during an experiment. Black circle in the center of the node represents physical location of a node. The effect of MDGA on NAC can easily be observed in the later snapshots, where a satisfactory separation among the nodes is obtained.

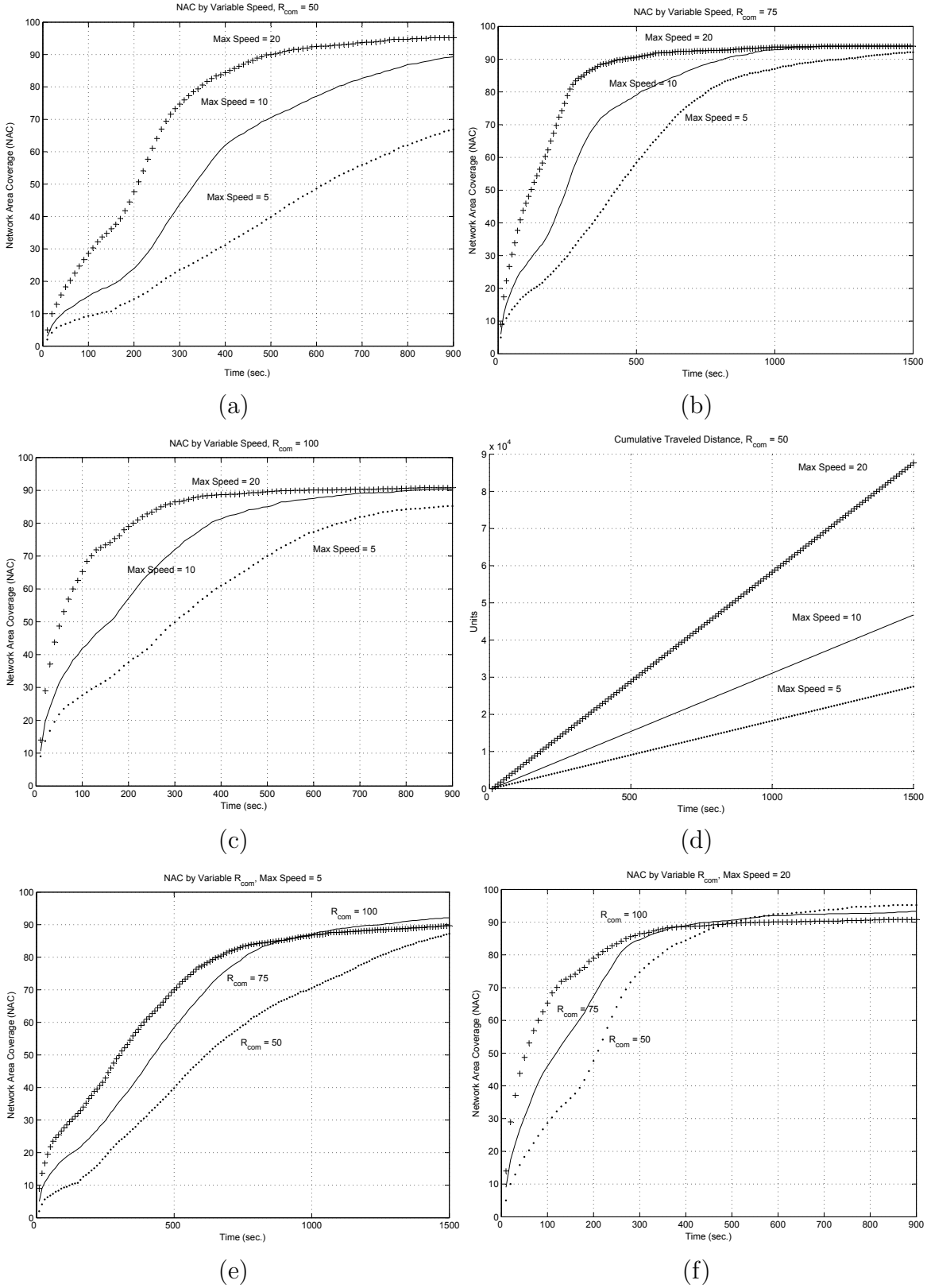


Figure 8.3: Effect of Speed on NAC for mobile nodes running MDGA in testbed.

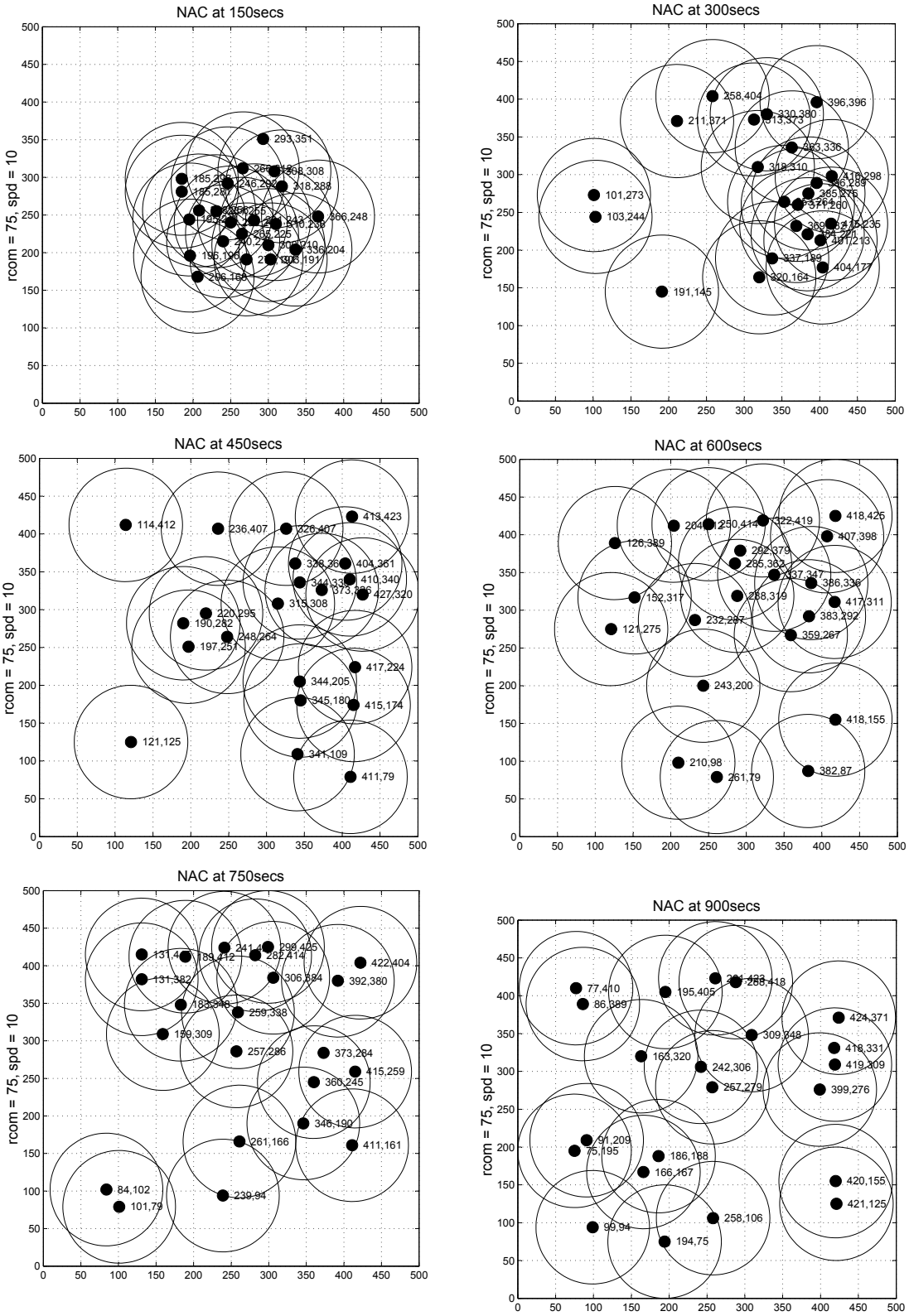
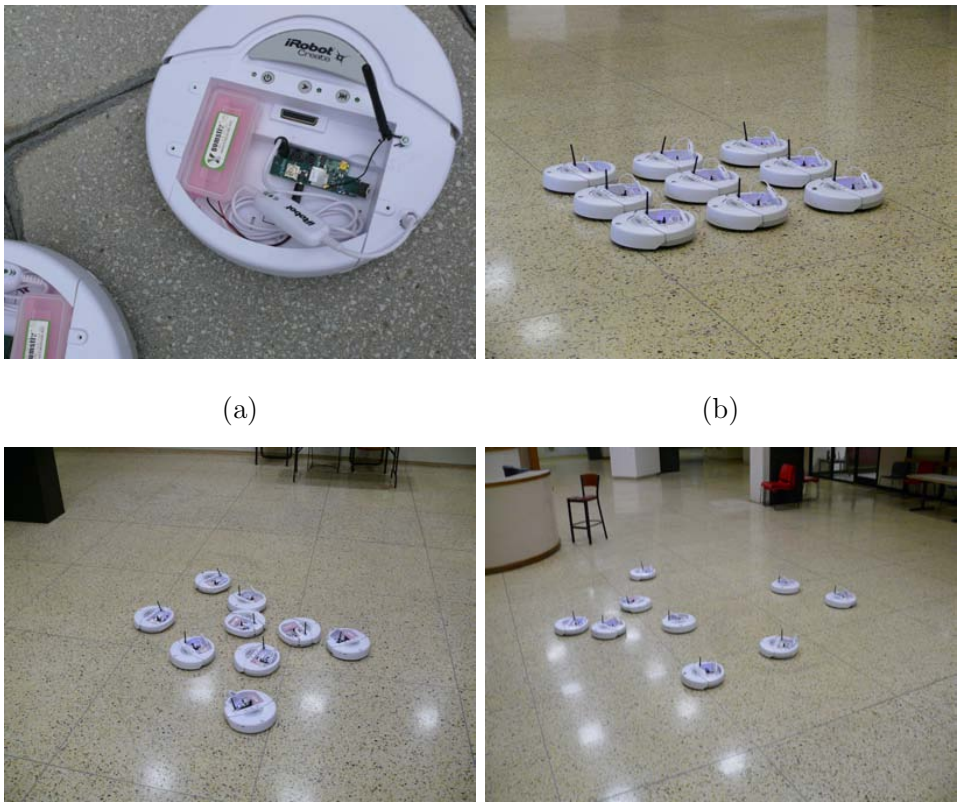


Figure 8.4: Self spreading of mobile nodes running MDGA in testbed, $R_{com} = 75$, speed = 10.



(a) (b)

(c) (d)

Figure 8.5: Node spreading experiments using *iRobots* controlled by the *Gumstix* processors with wireless capabilities (a total of 30 time units elapsed).

8.2 iRobot Testbed

iRobot Create robots with on-board Gumstix computers constitute the mobile nodes in this testbed implementation Fig. 8.5 (a). Gumstix computers are small in size, Linux-based, and consume very little power (<120mA @5V). The controller for the *iRobot* node is the Gumstix computer which runs our FGA, mobility commands including speed and direction, and wireless communication software. The nodes communicate with each other using Wi-Fi using UDP broadcast [68]. Our testbed has nine real integrated single-board computers (Gumstix) and automated robots (iRobots) as

shown in Figs. 8.5(b)-(d). The initial positions of the units are displayed in Fig. 8.5(b). Figs. 8.5(c) and (d) show two snapshots of movements of small robotic units by using our FGA.

8.3 Virtual Machine Testbed

We implemented a testbed using VMware technology, where several virtual machines (VMs) are run simultaneously on the same physical hardware by isolating each one from the physical environment by using VMware virtualization technique [62, 69].

For simplicity, the autonomous mobile nodes in our testbed are configured with the same capabilities, emulating realistic node mobility and wireless features of MANETs including, but not limited to, autonomous mobility, wireless communication characteristics, and periodic heartbeat messages (periodically broadcast to neighboring nodes within the communication range of R_{com} distance).

This testbed is programmed in C++ and runs in Windows, Linux, and Windows Mobile operating systems. Each computer in our testbed has a configurable number of VMs interconnected by a virtual switch, simplifying and allowing our mobile nodes running GA-based framework experiments on a single computer as though they run individually on a real network.

All VMs are connected to the network through a virtual switch. Typically, a single computer can handle approximately seven nodes for FGA applications. Each virtual

machine and the host have assigned addresses on the private network. This is done through the DHCP server included with the VMware Workstation.

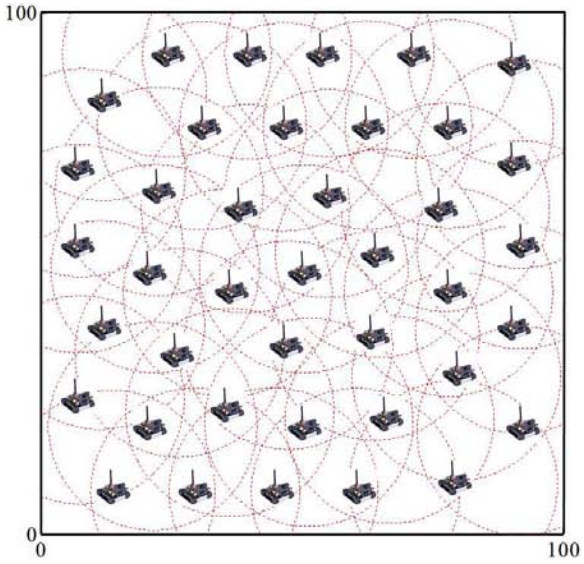


Figure 8.6: A screen shot of final mobile node distribution after 300 time units for mobile nodes spreading experiments using virtual machines.

In our testbed, we ran the experiments for a MANET with $N = 40$ mobile nodes and $R_{com} = 20$ for total of $T = 300$ time units in a terrain of 100 x 100 units. At the beginning of each experiment, all mobile nodes are located at the northwest corner of the given area (this is similar to the simulation software experiment given in Fig. 7.2). For simplicity, the mobile nodes in our testbed are configured with the same capabilities. The mobile node distribution after 300 time units is shown in Fig. 8.6. We can observe that, in spite of the lack of global knowledge and a centralized controller, the mobile nodes using our FGA obtain an almost uniform coverage of the area in a relatively short period of time.

In Fig. 8.7, the improvement in the effectiveness in NAC through time is shown for three different communication ranges of $R_{com} = 10$, 15, and 20 as the mobile

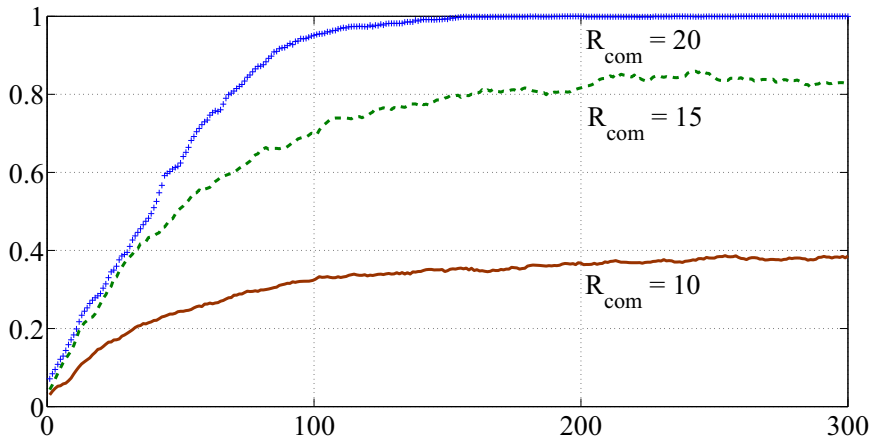


Figure 8.7: Effectiveness in NAC for mobile nodes spreading experiments using virtual machines.

nodes perform our GA-based topology control approach in this testbed. As seen from Fig. 8.7, the mobile nodes successfully deploy themselves in an unknown area and achieve 38%, 82%, and 100% area coverage for R_{com} = 10, 15, and 20, respectively. The area coverage reaches and stays at each maximum value at $T \approx$ 110, 160, and 200 time unit for testbed using R_{com} = 10, 15, and 20, respectively.

8.4 Laptops and PDAs

We implemented a testbed for studying the performance of our FGA using off-the-shelf laptops and personal digital assistants (PDAs) [70]. Each laptop and PDA runs an identical FGA topology control algorithm, which is taken from the virtual machine testbed. In Figs. 8.8 (a)-(d), each student has either a laptop or PDA which provides vocal and visual commands for direction and number of steps (emulating different speeds) using local neighborhood information. Fig. 8.8 (a) shows the initial deployment of nodes for this experiment where all students are placed together at

(a) (b)

(c) (d)

Figure 8.8: Node spreading experiments using laptops and PDAs (a total of 30 time units elapsed).

the bottom-right part of the area. The convergence towards a uniform distribution is displayed in Figs. 8.8 (b)-(d) over 30 time units.

Chapter 9

Conclusions

In this research, we introduce a GA-based topology control approach for efficient, reliable, and effective self-deployment of mobile nodes in MANETs. We implemented our bio-inspired algorithms for knowledge sharing mobile agents to obtain a uniform distribution of the nodes over an unknown terrain without any prior information or a centralized control unit. The shared knowledge includes the mobile agent location, speed, direction and degree. Each autonomous mobile node in a MANET runs our bio-inspired algorithms to control its speed and direction using only local neighborhood information.

We developed a simulation software in Java and implemented various testbed platforms to demonstrate the applicability of our approach to real-life problems. Simulation and testbed experiment results indicate that, for important performance metrics such as normalized area coverage (NAC), convergence rate, deployment time, and

average traveled time, Our GA-based topology control algorithm, called FGA, can be an effective mechanism to deploy nodes under restrained communication conditions in MANETs operating in unknown areas.

Since FGA adapts to the local environment rapidly and does not require global network knowledge, it can be used as a real-time topology controller for realistic military and civilian applications. Experiments results from our simulation software and testbed implementations showed that FGA delivers promising results for uniform node distribution of knowledge sharing mobile nodes over unknown geographical areas in MANETs.

We used a dynamical system model to formally study the convergence trajectory in the space of possible solutions of FGA and to analyze the effect of the initial population in the GA. We examined the expected behavior of our GA-based topology control approach over multiple generations and the cumulative effects of GA operators as a population evolves through several generations. We formally proved that, using FGA, a node moves to position with a better fitness (i.e., a position with a higher value of area coverage).

We analytically analyzed the convergence properties of our FGA topology control algorithm by modeling the network with a Markov chain model. Each state in the Markov chain represents different node configurations in a MANET with N nodes. The model demonstrates that FGA will both converge by analyzing Dobrushin's contraction coefficient and that that convergence will most likely be an optimal solution

by analyzing the total fitness of the final stationary distribution. It analyzes these two metrics among varied communication ranges and geographic areas. FGA performs very well and is only based on local information.

Following publications were generated during the course of this research.

Refereed Journal Papers:

1. C. S. Sahin, S. Gundry, E. Urrea, and M. U. Uyar, "Bio-inspired algorithms - Homogeneous Markov chain analysis on self-organizing MANETs," IEEE Transactions in Evolutionary Computations, (in review).

2. C. S. Sahin, E. Urrea, and M. U. Uyar, "Self Organization for Area Coverage Maximization and Energy Conservation in MANETs," Springer Transaction in Computer Science, Special Issue on Advances in Autonomic Computing: Formal Engineering Methods for Nature-Inspired Computing Systems (in press).

3. J. Kusyk, E. Urrea, C.S. Sahin, and M.U. Uyar, "Game Theory and Genetic Algorithm Based Approach for Self Positioning of Autonomous Nodes," Ad Hoc & Sensor Wireless Networks (in press)

4. J. Kusyk, C. S. Sahin, M. U. Uyar, E. Urrea, and S. Gundry, "Self Organization of Nodes in Mobile Ad Hoc Networks Using Evolutionary Game," Journal of Applied Research (invited paper) (in press).

5. C. S. Sahin, E. Urrea, M. U. Uyar, M. Conner, G. Bertoli, and C. Pizzo, "Design of Genetic Algorithms for Topology Control of Unmanned Vehicles," Interna-

tional Journal of Applied Decision Sciences, Special Issue on Decision Support Systems for Unmanned Vehicles 2010 (in press).

6. E. Urrea, C. S. Sahin, I. Hökelek, M. U. Uyar, M. Conner, G. Bertoli, and C. Pizzo, "Bio-inspired Topology Control for Knowledge Sharing Mobile Agents," Mobile Ad Hoc Networks, Elsevier, Special Issue on Bio-Inspired Computing, Vol. 7, No. 4, pp. 677-689, 2009.

Refereed Book Chapters:

1. C. S. Sahin, S. Gundry, E. Urrea, and M. U. Uyar, "A Bio-Inspired Approach to Self-organization of Mobile Nodes in Real-time Mobile Ad Hoc Network Applications," Book Chapter for Variants of Evolutionary Algorithms for Real-World Applications, Springer (in press).

2. C. S. Sahin, E. Urrea, and M. U. Uyar, "Bio-Inspired Algorithms for Self-organization in MANETs," IGI Book Chapter for Biologically Inspired Communications Networks (in press).

3. C. S. Sahin, E. Urrea, and M. U. Uyar, "Decentralized Topology Control for Autonomous Mobile Agents in MANETs," Book Chapter for Formal and Practical Aspects of Autonomic Computing and Networking: Specification, Development and Verification, IGI Global (in press).

Refereed Conference Publications:

1. E. Urrea, C. S. Sahin, M. U. Uyar, M. Conner, G. Bertoli and C. Pizzo, "Estimating Behavior of a GA-based Topology Control Mechanism for Self-Spreading Nodes in MANETs," IEEE Intl. Conf. on Military Communications (MILCOM) (in press)

2. J. Kusyk, E. Urrea, C. S. Sahin, M. U. Uyar, "Resilient Node Self-positioning Methods for MANETs Based on Game Theory and Genetic Algorithms," IEEE Intl. Conf. on Military Communications (MILCOM) (in press).

3. C. S. Sahin, S. Gundry, E. Urrea, M. U. Uyar, M. Conner, G. Bertoli and C. Pizzo, "Convergence Analysis of Genetic Algorithms for Topology Control in MANETs," 33rd IEEE Sarnoff Symposium, April 2010.

4. J. Kusyk, M. U. Uyar, E. Urrea, C. S. Sahin, "Game Theory Based Autonomous Mobile Nodes Distribution in MANETs," 33rd IEEE Sarnoff Symposium, April 2010.

5. C. S. Sahin, S. Gundry, E. Urrea, M. U. Uyar, M. Conner, G. Bertoli and C. Pizzo, "Markov Chain Models for Genetic Algorithm Based Topology Control in MANETs," Applications of Evolutionary Computation, EvoApplications 2010 (EvoComNet), pp. 41-50, April 2010.

6. J. Kusyk, M. U. Uyar, E. Urrea, C. S. Sahin, M. A. Fecko, and S. Samtani, "Efficient Node Distribution Techniques in Mobile Ad Hoc Networks Using Game Theory," IEEE Intl. Conf. on Military Communications (MILCOM 2009), pp. 1-7, October 2009.

7. C. Dogan, C. S. Sahin, M. U. Uyar, and E. Urrea, "Testbed for Node Communication in MANETs to Uniformly Cover Unknown Geographical Terrain Using Genetic Algorithms," The NASA/ESA Conference on Adaptive Hardware and Systems (AHS09), pp.273-280, San Francisco, CA, July 2009.

8. J. Kusyk and M. U. Uyar and E. Urrea and M. Fecko and S. Samtani, "Applications of game theory to mobile ad hoc networks: node spreading potential game," IEEE Sarnoff Symposium, pp. 1-5, March 2009.

9. C. S. Sahin, E. Urrea, M. U. Uyar, M. Conner, I. Hökelek, G. Bertoli and C. Pizzo, "Self-deployment of Mobile Agents in MANETs for Military Applications," Army Science Conference, pp. 1-8, Dec 2008.

10. C. S. Sahin, E. Urrea, M. U. Uyar, M. Conner, I. Hökelek, G. Bertoli and C. Pizzo, "Uniform Distribution of Mobile Agents Using Genetic Algorithms for Military Applications in MANETs," IEEE Intl. Conf. on Military Communications (MILCOM), pp.10-16, Nov 2008.

11. C. S. Sahin, E. Urrea, M. U. Uyar, M. Conner, I. Hökelek, G. Bertoli and C. Pizzo, "Genetic Algorithms for Self-Spreading Nodes in MANETs," The 10th Annual Conference on Genetic and Evolutionary Computation (GECCO), pp. 1141-1142, July 2008.

12. E. Urrea, C. S. Sahin, M. U. Uyar, M. Conner, I. Hökelek, G. Bertoli and C. Pizzo, "Comparative Evaluation of Genetic Algorithms for Force-Based Self-Deployment of Mobile Agents in MANETs," Int'l. Conf. on Genetic and Evo-

lutionary Methods, GEM'08/WorldComp'08, CSREA Press, USA, pp. 90-95, July 2008.

13. C. Dogan, M. U. Uyar, E. Urrea, C. S. Sahin, I. Hökelek, "Testbed Implementation of Genetic Algorithms for Self Spreading Nodes in MANETs," WorldComp/Gem'08, pp. 10-16, July 2008.

14. E. Urrea, C. S. Sahin, M. U. Uyar, M. Conner, H. Sharif, I. Hökelek, and G. Bertoli, "Simulation Experiments for Knowledge Sharing Agents Using Genetic Algorithms in MANETs," In Proc. Int'l. Conf. on Artificial Intelligence and Pattern Recognition (AIPR 07), pp. 369-376, July 2007.

15. E. Urrea, C. S. Sahin, M. U. Uyar, M. Conner, H. Sharif, I. Hökelek, G. Bertoli, and C. Pizzo, "Uniform MANET Node Distribution for Mobile Agents Using Genetic Algorithms," 2007 Int'l. Conf. on Genetic and Evolutionary Methods - (GEM 07), pp. 24-30, July 2007.

Bibliography

[1] I. Hökelek, M. Uyar, and M. Fecko, "Node link stability in wireless mobile networks," in *Defense Transformation and Network-Centric Systems*, ser. Proc. Society of Photo-Optical Instrumentation Engineers (SPIE), vol. 6249, Jun 2006.

[2] X. Inc., "Ml310 user guide, virtex-ii pro embedded development platform," February 1, 2007. [Online]. Available: www.xilinx.com/support/documentation/boards_and_kits/ug068.pdf

[3] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning.* Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1989.

[4] J. H. Holland, *Adaptation in Natural and Artificial Systems.* New York, NY: University of Michigan, 1975.

[5] L. Bayindir and E. Şahin, "A review of studies in swarm robotics," *Turk J Elec Engin*, vol. 15, no. 2, 2007.

[6] R. Zhang, Z. Wang, and J. Xiong, "Research on coordination of multi-robots system based on swarm intelligence," *Intelligent Systems Design and Applications.*

ISDA '06. Sixth International Conference on, vol. 2, pp. 653–660, Oct. 2006.

[7] C. S. Sahin, S. Gundry, E. Urrea, M. U. Uyar, M. Conner, G. Bertoli, and C. Pizzo, "Markov chain models for genetic algorithm based topology control in manets," in *EvoApplications (2)*, ser. Lecture Notes in Computer Science. Springer, 2010, pp. 41–50.

[8] ——, "Convergence analysis of genetic algorithms for topology control in manets," in *Sarnoff Symposium, 2010 IEEE*, 12-14 2010, pp. 1 –5.

[9] Y. Hu and S. Yang, "A knowledge based genetic algorithm for path planning of a mobile robot," in *Robotics and Automation. Proceedings. ICRA '04. 2004 IEEE International Conference on*, vol. 5, Apr. 2004, pp. 4350–4355.

[10] M. Lee, S. Joo, and H. Kim, "Realization of emergent behavior in collective autonomous mobile agents using an artificial neural network and a genetic algorithm," *Neural Comput. Appl.*, vol. 13, no. 3, pp. 237–247, 2004.

[11] T. Shinchi, M. Tabuse, T. Kitazoe, and A. Todaka, "Khepera robots applied to highway autonomous mobiles," *Artificial Life and Robotics*, vol. 7, pp. 118–123, 2003.

[12] M. Chen and A. Zalzala, "Safety considirections in the optimization of the paths for mobile robots using genetic algorithms," in *1st Intl. Conf. on Genetic Algorithms in Engineering Systems: Innovations and Applications, GALESIA*, 1995.

[13] V. Gesu, B. Lenzitti, G. Bosco, and D. Tegolo, "A distributed architecture for autonomous navigation of robots," in *Proc. 5th IEEE Intl. Workshop on Computer Architectures for Machine Perception*, 2004.

[14] M. L. Pilat and F. Oppacher, "Robotic control using hierarchical genetic programming," in *Genetic and Evolutionary Computation GECCO 2004*, vol. 3103. Springer Berlin / Heidelberg, 2004, pp. 642–653.

[15] B. Bishop, F. Crabbe, and B. Hudock, "Design of a low-cost, highly mobile urban search and rescue robot," *Advanced Robotics, Brill Academic Publisher*, 2005.

[16] R. L. Dollarhide and A. Agah, "Simulation and control of distributed robot search teams," *Computers and Electrical Engineering*, vol. 29, no. 5, pp. 625–642, 2003.

[17] L. Moreno, J. M. Armingol, S. Garrido, A. de la Escalera, and M. A. Salichs, "A genetic algorithm for mobile robot localization using ultrasonic sensors," *Journal of Intelligent and Robotic Systems*, vol. 34, pp. 135–154, 2002.

[18] J. Werfel, "Robot search in 3d swarm construction," in *Self-Adaptive and Self-Organizing Systems. SASO '07. First International Conference on*, Jul. 2007, pp. 363–366.

[19] W. Li, Y. Liu, H. Deng, and Y. Xu, "Obstacle-avoidance path planning for soccer robots using particle swarm optimization," in *Proc. IEEE Intl. Conf. on Robotics and Biomimetics*, 2006, pp. 1233–1238.

[20] S. Li, X. Sun, and Y. Xu, "Particle swarm optimization for route planning of unmanned aerial vehicles," in *Proc. IEEE Intl. Conf. on Information Acquisition*, 2006, pp. 1213–1218.

[21] F. Weicing, W. KeJun, Y. XiuFen, and G. Shuxiang, "Novel algorithms for coordination of underwater swarm robotics," in *Proc. IEEE Intl. Conf. on Mechatronics and Automation*, 2006, pp. 654–659.

[22] J. Zhao, J. Jiang, and X. Zang, "Cooperative multi-robot map-building based on genetic algorithms," in *Proc. IEEE Intl. Conf. on Information Acquisition*, 2006, pp. 360–364.

[23] A. Howard, M. Mataric, and G. Sukhatme, "Mobile sensor network deployment using potential fields: A distributed, scalable solution to the area coverage problem," in *Proceedings of the 6th International Symposium on Distributed Autonomous Robotics Systems (DARS02).*, Fukuoka, Japan, June 2002.

[24] A. F. Winfield, "Distributed sensing and data collection via broken ad hoc wireless connected networks of mobile robots," *Distributed Autonomous Robotic Systems 4, Springer-Verlag*, pp. 273–282, 2000.

[25] N. Heo and P. Varshney, "A distributed self spreading algorithm for mobile wireless sensor networks," in *Wireless Communications and Networking, IEEE*, vol. 3, Mar. 2003, pp. 1597–1602.

[26] M. Ma and Y. Yang, "Adaptive triangular deployment algorithm for unattended mobile sensor networks," *Computers, IEEE Transactions on*, vol. 56, no. 7, pp. 946–847, Jul. 2007.

[27] F. Bai and A. Helmy, *A Survey of Mobility Modeling and Analysis in Wireless Adhoc Networks*. Berlin, Germany: Springer, 2006.

[28] G. Lin, N. G, and R. Rajaraman, "Mobility models for ad hoc network simulation," in *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, vol. 1, Jul. 2004, p. 463.

[29] T. Camp, J. Boleng, J. Boleng, and V. Davies, "A survey of mobility models for ad hoc network research," in *Wireless Communications and Mobile Computing*, 2002, pp. 483–502.

[30] I. Hökelek, M. Uyar, and M. Fecko, "A novel analytic model for virtual backbone stability in mobile ad hoc networks," *Wireless Networks, Springer*, vol. 6249, no. 14, pp. 87–102, 2008.

[31] I. Hökelek, "Analytic models and distributed robotics applications for mobile ad hoc networks," Ph.D. dissertation, City University of New York Graduate Center, New York, NY, 2006.

[32] Y. Shang and S. Cheng, "A stable clustering formation in mobile ad hoc network," in *Wireless Communications, Networking and Mobile Computing, Proceedings. 2005 International Conference on*, vol. 2, Sep. 2005, pp. 714–718.

[33] D. Turgut, S. Das, R. Elmasri, and B. Turgut, "Optimizing clustering algorithm in mobile ad hoc networks using genetic algorithmic approach," *Global Telecommunications Conference, 2002. GLOBECOM '02. IEEE*, vol. 1, pp. 62–66, 17-21 Nov. 2002.

[34] W. Jin, X. Li, and Z. Baoyu, "A genetic annealing hybrid algorithm based clustering strategy in mobile ad hoc network," in *Communications, Circuits and Systems, Proceedings. 2005 International Conference on*, vol. 1, May. 2005, pp. 314–318.

[35] L. Barolli, A. Koyama, and N. Shiratori, "A qos routing method for ad-hoc networks based on genetic algorithm," in *Database and Expert Systems Applications. Proceedings. 14th International Workshop on*, Sep. 2003, pp. 175–179.

[36] D. Montana and J. Redi, "Optimizing parameters of a mobile ad hoc network protocol with a genetic algorithm," in *GECCO '05: Proceedings of the 2005 conference on Genetic and evolutionary computation.* New York, NY, USA: ACM, 2005, pp. 1993–1998.

[37] C. Ahn and R. Ramakrishna, "A genetic algorithm for shortest path routing problem and the sizing of populations," *Evolutionary Computation, IEEE Transactions on*, vol. 6, no. 6, pp. 566–579, Dec 2002.

[38] X. Ma, Q. Zhang, and Y. Li, "Genetic algorithm-based multi-robot cooperative exploration," in *IEEE Int. Conference on Control and Automation*, Jun. 2007, pp. 1018–1023.

[39] A. E. Nix and M. D. Vose, "Modeling genetic algorithms with markov chains," *Annals of Mathematics and Artificial Intelligence*, vol. 5, pp. 79–88, 1992.

[40] M. D. Vose, *The Simple Genetic Algorithm: Foundations and Theory.* Cambridge, MA, USA: MIT Press, 1998.

[41] J. Suzuki, "A markov chain analysis on simple genetic algorithms," *Systems, Man and Cybernetics, IEEE Transactions on*, vol. 25, no. 4, pp. 655–659, Apr. 1995.

[42] T. Nakama, "Transition and convergence properties of genetic algorithms applied to fitness functions perturbed concurrently by additive and multiplicative noise," in *Evolutionary Computation. CEC '09. IEEE Congress on*, May. 2009, pp. 2662–2669.

[43] L. Ming, Y. Wang, and Y.-M. Cheung, "On convergence rate of a class of genetic algorithms," *World Automation Congress, 2006. WAC '06*, pp. 1–6, 24-26 July 2006.

[44] H. Aytug, S. Bhattacharrya, and G. J. Koehler, "A markov chain analysis of genetic algorithms with power of 2 cardinality alphabets," *European Journal of Operational Research*, vol. 96, no. 1, pp. 195 – 201, 1997.

[45] J. Kusyk, M. U. Uyar, E. Urrea, M. A. Fecko, and S. Samtani, "Applications of game theory to mobile ad hoc networks: node spreading potential game," in *Proceedings of the 32nd international conference on Sarnoff symposium.* Piscataway, NJ, USA: IEEE Press, 2009, pp. 65–69.

[46] O. Khatib, "Real-Time Obstacle Avoidance for Manipulators and Mobile Robots," *The International Journal of Robotics Research*, vol. 5, no. 1, pp. 90–98, 1986.

[47] M. Mitchell, *An Introduction to Genetic Algorithms*. Cambridge, MA, USA: MIT Press, 1998.

[48] E. Urrea, C. S. Sahin, I. Hökelek, M. U. Uyar, M. Conner, G. Bertoli, and C. Pizzo, "Bio-inspired topology control for knowledge sharing mobile agents," *Ad Hoc Netw.*, vol. 7, no. 4, pp. 677–689, 2009.

[49] E. Urrea, C. S. Sahin, M. U. Uyar, I. Hökelek, M. Conner, G. Bertoli, H. Sharif, and C. Pizzo, "Uniform manet node distribution for mobile agents using genetic algorithms," in *Proc. International Conference on Genetic and Evolutionary Methods (GEM)*, 2007, pp. 24–30.

[50] C. S. Sahin, U. Urrea, M. U. Uyar, M. Conner, I. Hökelek, G. Bertoli, and C. Pizzo, "Genetic algorithms for self-spreading nodes in manets," in *GECCO '08: Proceedings of the 10th annual conference on Genetic and evolutionary computation*. New York, NY, USA: ACM, 2008, pp. 1115–1116.

[51] E. Urrea, C. S. Sahin, M. U. Uyar, M. Conner, I. Hökelek, G. Bertoli, and C. Pizzo, "Comparative evaluation of genetic algorithms for force-based self-deployment of mobile agents in manets," in *Proc. International Conference on Genetic and Evolutionary Methods (GEM)*, 2008, pp. 90–85.

[52] I. Hökelek, M. U. Uyar, and M. A. Fecko, "Random-walk based analysis of virtual backbone in manets," in *Proc. of 2005 Communications and Computer Networks*, 2005, pp. 132–137.

[53] E. Urrea, C. S. Sahin, M. U. Uyar, M. Conner, H. Sharif, I. Hökelek, and G. Bertoli, "Simulation experiments for knowledge sharing agents using genetic algorithms in manets," in *Artificial Intelligence and Pattern Recognition*, 2007, pp. 369–376.

[54] G. Winkler, *Image Analysis, Random Fields and Markov Chains Monte Carlo Methods*. Springer-Verlag Berlin Heidelberg, 2006.

[55] W.-K. Ching, M. K. Ng, and W. Ching, *Markov Chains: Models, Algorithms and Applications (International Series in Operations Research & Management Science)*. Springer, Dec. 2005.

[56] A. A. Borovkov, *Ergodicity and Stability of Stochastic Processes*. Wiley, Sep. 1998.

[57] J. Hachigian, "Collapsed markov chains and the chapman-kolmogorov equation," *The Annals of Mathematical Statistics*, vol. 34, no. 1, pp. 233–237, Mar. 1963.

[58] R. Dobrushin, "Central limit theorem for nonstationary markov chains. ii," *Teor. Veroyatnost Primenen*, pp. 365–425, 1956.

[59] C. S. Sahin, E. Urrea, M. U. Uyar, M. Conner, I. Hökelek, G. Bertoli, and C. Pizzo, "Self-deployment of mobile agents in manets for military applications," in *Army Science Conference*, 2008, pp. 1–8.

[60] *iRobot developer and educators site*, store.irobot.com/deved/index.jspl.

[61] *Gumstix developer site*, Users manual, www.gumstix.net/Documentation/109.html.

[62] *Workstation Users Manual, Workstation 6.5*, VMware Inc., 2008.

[63] C. Dogan, M. Ü. Uyar, E. Urrea, C. S. Sahin, and I. Hökelek, "Testbed implementation of genetic algorithms for self spreading nodes in manets," in *Proc. International Conference on Genetic and Evolutionary Methods (GEM)*, 2008, pp. 10–16.

[64] *VxWorks Programmer's Guide*, Wind River Systems, Inc, 1993.

[65] X. Inc., "System ace compactflash solution," April 5, 2002. [Online]. Available: www.xilinx.com/support/documentation/data_sheets/
ds080.pdf

[66] ——, "Powerpc 405 processor block reference guide, embedded development kit," June 5, 2007. [Online]. Available: www.xilinx.com/support/documentation/user_guides/ug018.pdf

[67] I. Wind River Systems, "Vxworks programmer's guide 5.5," 2003. [Online]. Available: www-cdfonline.fnal.gov/daq/commercial/
vxworks_programmers_guide5.5.pdf

[68] C. Dogan, C. Sahin, M. Uyar, and E. Urrea, "Testbed for node communication in manets to uniformly cover unknown geographical terrain using genetic algo-

rithms," *Adaptive Hardware and Systems. AHS 2009. NASA/ESA Conference on*, pp. 273–280, Jul. 2009.

[69] R. J. Barnett and B. Irwin, "Performance effects of concurrent virtual machine execution in vmware workstation 6," in *Advanced Techniques in Computing Sciences and Software Engineering*. Springer Netherlands, 2010, pp. 329–333.

[70] M. Parkin and J. M. Brooke, "A pda client for the computational grid," *Concurrency and Computation: Practice and Experience*, vol. 19, no. 9, pp. 1317–1331, 2007.