FINAL REPORT

Colorado Advanced Software Institute

RACEWIN:
Resource Allocation and Admission Control Evaluator for Wireless Information Networks

Principal Investigator:	C. Edward Chow, Ph.D. Associate Professor University of Colorado, Colorado Springs
Graduate Students:	Heidi McClure and Jonas Hedlind Department of Computer Science University of Colorado, Colorado Springs
Collaborating Company:	US WEST Advanced Technologies Vicky Okeson Member of Technical Staff

PROJECT TITLE: RACEWIN: Resource Allocation and Admission Control Evaluator for Wireless Information Networks

PRINCIPAL INVESTIGATOR: C. Edward Chow, Ph.D.

UNIVERSITY: University of Colorado, Colorado Springs

COLLABORATING COMPANY: US WEST Advanced Technologies

REPRESENTATIVE OF COLLABORATING COMPANY: Vicky Okeson, Member of Technical Staff

As authorized representative of the collaborating company, I have reviewed this report and approve it for release to the Colorado Advanced Software Institute.

SIGNATURE:_____

DATE:_____

Table of Contents

Abstract

This project deals with the development of efficient algorithms and simulation tools for PCS Wireless Information Network Planning and Management. A software design system called Resource Allocation and Admission Control Evaluator for Wireless Information Networks (RACEWIN) was built to facilitate the design and analysis of efficient algorithms for cell size determination, power and cell site assignment, and traffic management in the future wireless information networks, such as the emerging PCS networks. The RACEWIN system will assist the network administrators in their network planning and management tasks. It will facilitate the network designers to improve the network efficiency and reliability.

RACEWIN provides a Java-based user traffic modeling and simulation tool called UTMOST and a power and cell site assignment tool called POCAT. UTMOST allows the simulation of different user traffic patterns on the selected 4-cell high way or 7-cell stadium network model. It generates event records for other optimization packages and discrete event simulators. The records include the user locations, power and cell site assignments, boundary crossing times, and calling status changes. UTMOST provides an enhanced graphical user interface for displaying and verifying power and cell site assignments. It also allows the graphical display of user traffic simulation steps both forward and backward. The power assignments of individual users are displayed in color code and the cell site assignments as links between the users and the base stations. The animation effect helps the understanding and identification of the problems in power and cell site assignment procedures.

POCAT implemented a power and channel assignment algorithm that reduces the total interference by minimizing the overall transmitting power of users. It takes the location data generated by UTMOST as input and considers the impact of the overlap angle among the sector antennae. A C-version of POCAT can interface with UTMOST through file interface and the Java-based version of POCAT is integrated with UTMOST.

By integrating with discrete event simulators which model the wireless network resources, RACEWIN can be used for studying the QoS of a wireless network under different traffic loads and patterns, and for the cell size determination. By providing traffic data to optimization packages for power and cell site assignments and by graphically displaying and verifying the results generated, RACEWIN can help improve those optimization packages. RACEWIN can serve as an important module for wireless information network planning and management and help improve the efficiency and reliability of wireless information networks.

1. Introduction

Personal Communications Service (PCS) has been referred to as a concept that will make it possible to communicate with anyone-anytime-anywhere. The FCC has defined PCS as "radio communications that encompass mobile and ancillary fixed communications that provide services to individuals and businesses and can be integrated with a variety of competing networks." [1] Furthermore, the FCC characterized PCS as encompassing "a broad range of new radio communications service that will free individuals from the limitations the wireline public switched telephone network and will enable individuals to communicate when they are away from their home or office telephones." [2] In 1994, the PCIA predicted that there will be 167 million subscriptions to PCS services in the United States by 2003 with many users subscribing to multiple wireless services [3].

Since the radio spectrum is limited, future wireless systems will have macro/micro/picocellular architectures in order to provide the higher capacity needed to support higher number of users and wide range of services from narrowband voice to broadband multimedia applications. The design of the future multi-tier cellular networks involves the selection of cell size at each tier and the design of efficient admission control, bandwidth assignment and handoff procedures. Each is a challenging network design and management problem [4,5,6]. Reduced size coverage zones like micro and pico cells will offer higher traffic handling capability but will induce an increase in handover activity [7,8,9,10]. Distributed control may increase call handling capability and therefore the possibility of further cell size reduction. It is also a challenging research issue to make the design trade-off among the inter-related network parameters [6].

After discussed with researchers at US West Technologies earlier in the project, we have focused on the creation of a user traffic modeling and simulation tool for wireless information networks, since the traffic data are very important for the planning and evolution of wireless information networks. The traffic data are also needed for exercising and improving the power and channel assignment optimization packages that US West Technologies was working on. There is also a need for verifying the power and cell site assignment results generated by those optimization packages. Independently we developed a power and cell site assignment tool, called POCAT, which can be used to compare the power and cell site assignment results generated by those packages. The resulting RACEWIN system has realized these goals.

Section 2 discusses the objectives. Section 3 lists the approaches. Section 4 discusses the results. Section 5 is the evaluation. Section 6 summarizes the technology exchange between US West Advanced Technologies and UCCS. Appendices include the man pages, the file formats, and the user guide of the RACEWIN system.

2. Objectives

The general objective of the proposed research is to develop a set of network design tools and algorithms for the design and analysis of wireless networks that utilize network resources efficiently and reliably.

First, we propose to extend our existing literature survey for the related works on wireless information network design and traffic management. We will define network parameters to be considered in the wireless information network design and traffic management problems. The parameters include at least: 1) network topology, 2) cell size, 3) cell bandwidth, 3) traffic patterns, 4) processing power of base stations and switches, and 5) handover rate. We will also define a set of metrics for evaluating the wireless information network designs. They include the number of calls served per hour per cell, the bandwidth utilization of the system, the number of customers in the system, the response time, call blocking probability, handover blocking probability, and duration of interruption of user traffic.

Second, we propose to build network models that simulate different wireless network architectures. For each network model we will design and analyze the network design and management procedures. This research will be based upon work we have been doing on network simulation, optimization algorithms for spare usage in network restoration [12,13,14,15] and ATM resource allocation and traffic management [16,17].

Third, with the help of US West researchers, realistic traffic patterns will be modeled. An extensible network simulation system with an enhanced graphic user interface will be built to compare different PCS network designs and to facilitate the analysis of the handover rate, call/handover blocking probability, and service interrupt duration. RACEWIN will be integrated with GNUPLOT to facilitate the presentation of simulation results.

Fourth, we propose to evaluate different design trade-off of PCS networks. The research results will suggest promising PCS network design choices with excellent overall performance in terms of the network throughput and quality of services.

The algorithms and the network simulation prototype created in the proposed research will form a technology and knowledge base to enable and facilitate network designers to design efficient and reliable PCS networks. They will provide network administrators with efficient tools to plan or utilize more efficiently the available bandwidth in the future PCS networks and to provide reliable network services to the users.

As we indicated in the Introduction section, working with researchers at US West Advanced Technologies, we have narrowed our focus on the user traffic modeling and simulation tool, and the development of a power and cell site assignment tool.

3. Approach

The proposed project will proceed with the following four phases:

3.1 Definition Phase

Based on the survey of the general characteristics of PCS networks and our research on network design and simulation techniques, we will define a common terminology and a set of quantitative measures on the efficiency of network designs.

3.2 Modeling Phase

We will define a network model which includes different cell layout patterns and user traffic models. The cell layout and user traffic models should enable the studies of mobile traffic along a highway, where they are heavy in one direction in rush hours with fast handover rates, and those of special events, such as the football games, where the mobile traffic converges and disperses. Source traffic characteristics include the call originating time, roaming, call duration, route and speed of mobile users, peak/average bit rates, burst duration, and distribution of active and silent periods. The sector antennae, overlap angles, the capacity of the base stations should be included in the network model.

The following parameterized variables should be included in the model:

1. Cell Layout patterns including base station locations and boundaries.
2. User traversing patterns including speeds, directions, and boundary crossing times and places.
3. User calling patterns.
4. User power and antenna assignments including soft handoff sets of antennae.
5. Sector antenna directions and overlap angles.
6. The Capacity of the base station including the numbers of channels served.
7. Signal fading and interference.

3.3 Design and Analysis Phase

We will build a RACEWIN system based on the proposed network model. The simulation software modules will be implemented based on the performance measures established in Phase 1.

First, a graphical user interface program will be built for displaying the cell layout patterns, the current locations and power and cell site assignments of users. The power assignments will be color coded. The antenna assignments can be displayed as links between the mobile user and the assigned base stations. This will help verify the power and cell site assignment results from other modules. To facilitate the study of the transient behavior, the user can choose to display those links when their power or antenna assignments are changed.

The user traversing patterns will be simulated in sequence of steps. The simulation can be advanced in a single step or multiple steps. For each simulated step, data files will be generated which gives a snapshot of the current network and user status, including the

coordinate, current cell or sector, power and antenna assignment of each user. To facilitate the study of efficient handoff procedures, the boundary crossing times and places will be computed and saved in the data files. To facilitate the observation and analysis, the generated simulation files can be read and displayed in reverse order.

These simulation files need to be well organized to allow efficient access from other packages or simulators. The interface should be defined to enable the graphical user interface program to read in the power and cell site assignments generated by other packages. It should allow the flexibility of running these packages cooperatively on-line or off-line.

A power and cell site assignment tool based on the approaches of Steve Hanly [38], and Yates and Huang [39] will be implemented. A proper signal fading model will be chosen for calculating the power level. This tool will use the data generated by the user traffic modeling tool as input, and generate the power and cell site assignments generated to be read back by the graphical user interface.

3.4 Integration Phase

After constructing the efficient power and cell site assignment tool and the user traffic modeling and simulation tool as basic modules, we will investigate how they can best be integrated. We will study the performance of the power and cell site assignment tool given certain network design parameters. With the help of RACEWIN, we will explore the relationship among the network architectures, the cell sizes, the network control procedures, and the traffic patterns.

3.5 Detailed plan for the RACEWIN system

RACEWIN will be built using the algorithms, software libraries, and tools provided by the Computer-Aided Network Design & Analysis Research Environment, CANDARE, which was developed at UCCS and was used successfully to develop a generalized resource allocation system, RAS[16], and NETRESTORE [14]. CANDARE will facilitate us to construct the network models and the power and cell site assignment algorithms. We will design the proposed algorithms with the needed controllable parameters for modeling the capacity of base station, the overlap antenna angle, signal fading, and the number of uses and their traversing patterns. US West will involve in the design and implementation process, provide information about PCS network topologies and traffic patterns, and provide the important feedback to improve the accuracy, functionality and performance of the RACEWIN system.

4. Results

4.1 User Traffic Modeling and Simulation Tool (UTMOST)

UTMOST is the name of the tool. The acronym stands for User Traffic MOdeling and Simulation Tool. The tool has a Graphical User Interface (GUI) and is a stand-alone Java Application [27, 28, 29, 30, 31].

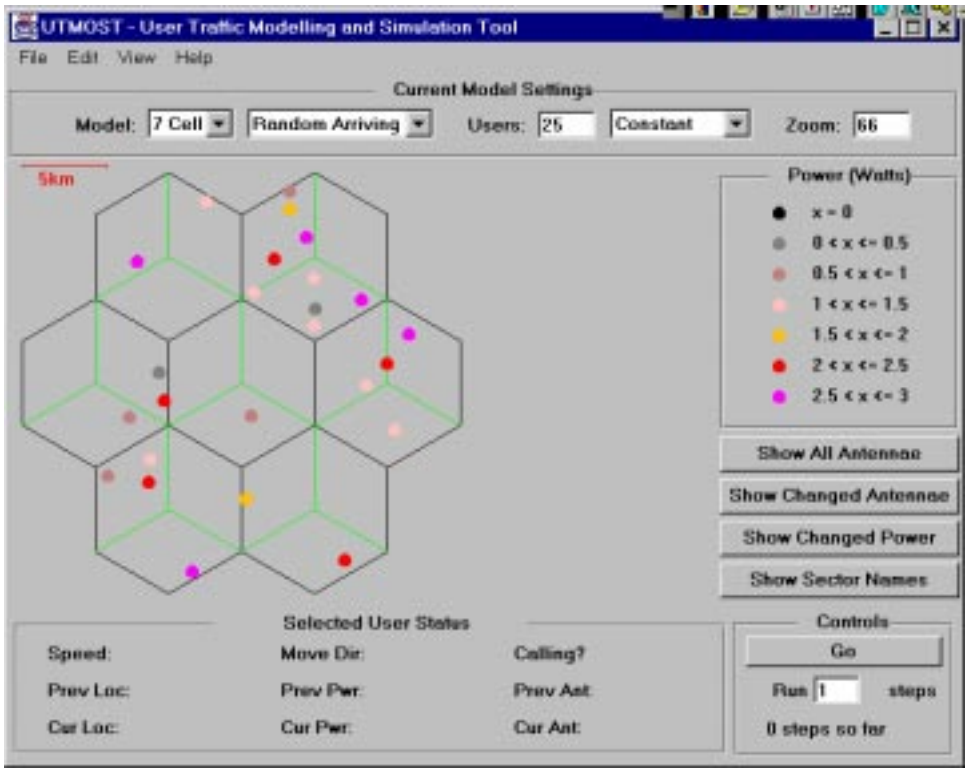


Figure 1. The UTMOST GUI

The tool shows users as colored dots on a map of sectors. A PCS cell is represented by a hexagon [17, 18, 24, 25, 37]. A sector is one third of a PCS cell. Each hexagon has at its center a base station antenna. This antenna is actually three sector antennae which are directionally aimed. Each sector antenna covers sectors of cells that falls within its 120 degree coverage area. Since the reception power from a mobile station is typically the inverse of its distance to the sector antenna raised to the power of four. Therefore the sectors that are too far from the sector antenna, the signals of a mobile station from those sectors will be lower than a threshold and be too weak to be detected. In this case, the sector antenna is considered to be *invisible* to the mobile station. Other covered sectors, that are closer to the sector with mobile station signal above the threshold, the sector antenna is said to be *visible* to the mobile station in the covered sectors.

Each cell has a radius. Currently, in UTMOST, the radius is five kilometers. The radius may be changed by the command line interface (-DRADIUS=4.3) and the network configuration file (setting coordinates at 4.3 radius increments). On the maps, the radius is the same as one of the lines radiating out from a base station (center of hexagon) to the edge of that hexagon.

4.1.1 What is It For? (the models)

UTMOST is a user traffic modeling and simulation tool intended for use with PCS Wireless Information Network (WIN) simulations. The WIN simulations are used to study power and cell assignment algorithms and optimizations. That is, using input data from UTMOST of where mobile users are located at discrete points in time, the WIN simulations would be able to study and optimize power assigned to each user and decide

if a new assignment is required. Similarly, channels to which users are assigned may need to change - a channel would be associated with a base station (antenna) and if a particular base station is over loaded, users may be handed off to a nearby antenna.

WIN simulations need to have accurate and realistic user traffic models as generated, for example, by UTMOST because actual PCS networks most commonly handle mobile users who travel from one cell to another, usually by automobile. The PCS WINs must try to optimize handling of mobile users so user calls are not disconnected. That is, WIN simulations are trying to help optimize and ultimately make more reliable all of the PCS networks.

UTMOST generates data of various traffic patterns. These traffic patterns could simulate mobile PCS users moving in automobiles or on foot in geographical regions. For example, along major interstate highways in the U. S., there are a number of antennae which provide PCS coverage. UTMOST would provide user traffic patterns simulating mobile users on the highway. The highway would have a number of sectors providing PCS communication. As users move about the cells and sectors, UTMOST calculates when a users has crossed from one sector to another. Boundary crossing information becomes important to the main switching stations in the WIN - the switching station must figure out if the user needs to be handed off from one antenna to another antenna. In addition to calculating the boundary crosses, UTMOST also provides information about which antennae in the model are visible to the mobile user. That is, since the antennae are directional, only one antenna in a cell can see a user at any time.

UTMOST provides some of the following specific modeling capabilities:

- From one to five thousand mobile users may be modeled.
- A circular seven cell layout and a linear four cell layout patterns are provided. The circular seven cell model simulates the events where mobile users gather at the central cell, e.g., a football stadium. The linear four cell model simulates highway traffic.
- Arriving, leaving and random patterns may be selected. The users may start randomly or may start from the perimeter or center of the model.
- Users may be constant in the model or they may regenerate after reaching their destination.

  Constant user behavior means that once a user is initialized, that user never changes its identity. That is, if the user travels outside the boundaries of the model, the user just continues on its path, or if the user reaches its destination in the center of the model, the user just stays at the center.

  Regenerating user behavior means that once a user reaches its destination, this user is re-initialized. That is, the user is given totally new coordinates, power levels, and direction of travel.

4.1.2 How is It Used? (brief user's tutorial)

UTMOST is a Java application, so it requires a Java Virtual Machine (JVM) in order to be executed. It has been successfully run on Microsoft's Win95 and WinNT, Sun's

Solaris, Digital's Ultrix and SGI's IRIX operating systems. It requires Java Developer's Kit (JDK) 1.0.2 or greater version of the JVM. To execute UTMOST, you must be in the directory in which the UTMOST class files are (~racewin/src) and you may need to set the Java CLASSPATH environment variable to include the current directory ("setenv CLASSPATH ." for Unix csh.) To run UTMOST type "java UTMOST". The window in Figure 1 should then appear.

Next you may wish to select other initial model settings than the defaults shown. Select the model pattern by pressing the first "Model:" pop-up menu and select "7 Cell" or "4 Cell" cell layout pattern:

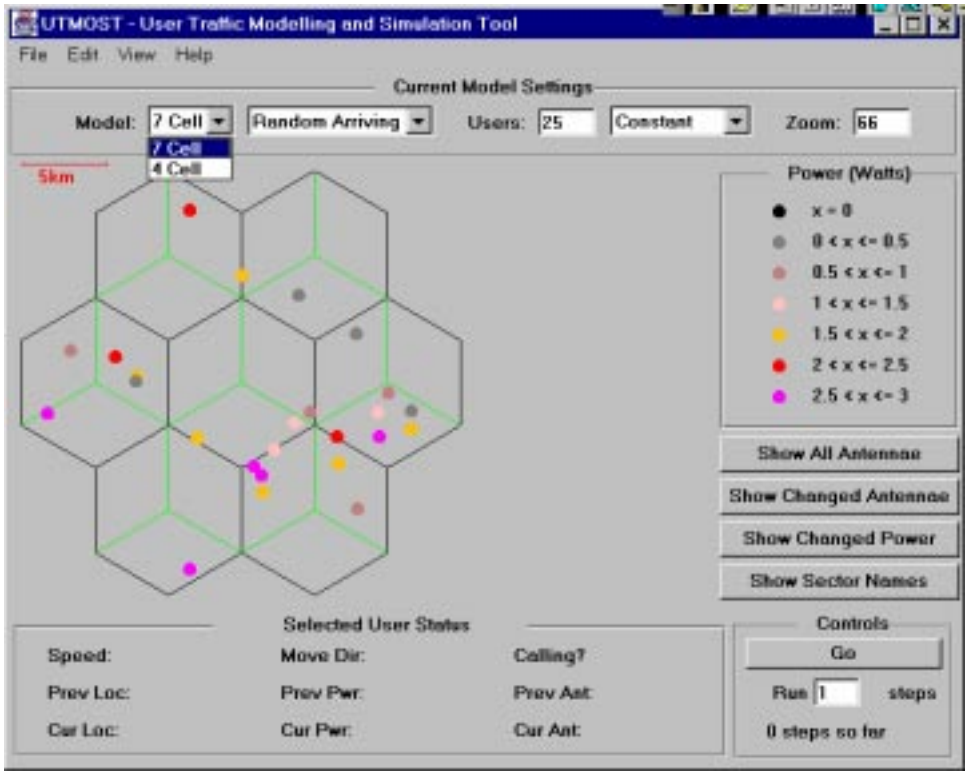


Figure 2. Model Pattern Selection Pop-up Menu

Now select the type of model you wish to run:

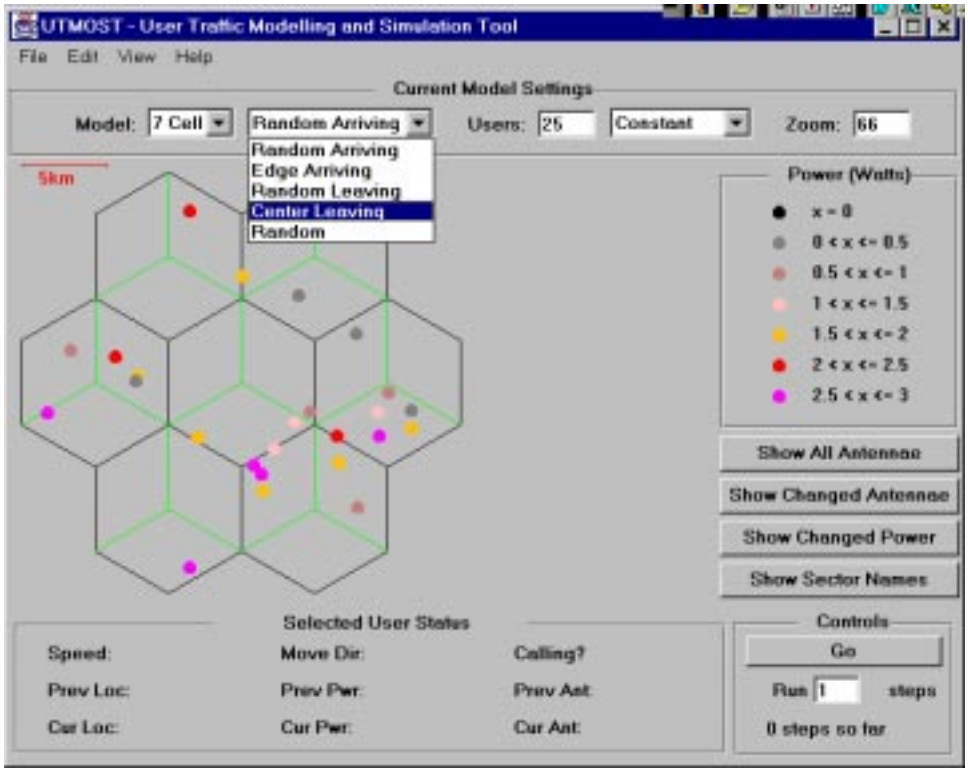


Figure 3. Model Type Selection Pop-up Menu

Table 1 shows the summary of all the pattern and model configurations which UTMOST will allow.

Table 1. UTMOST Model Types

Pattern	Model Type	Description
7 Cell	Random Arriving	Users start from random locations inside the 7 Cell pattern and all move towards the center.
7 Cell	Edge Arriving	Users start from the edge (perimeter) of the 7 Cell pattern and all move towards the center.
7 Cell	Random Leaving	Users start from random locations inside the 7 Cell pattern and all move towards the outside edge of the model in a straight line from the center.
7 Cell	Center Leaving	Users start from the center of the model and all move towards the outside edge of the model in a straight line from the center.
7 Cell	Random	Users start from random locations inside the 7 Cell pattern and move in random straight line directions.
4 Cell	Random Arriving	Users start from random locations on the horizontal center line of the 4 Cell pattern and all move towards the right.
4 Cell	Arriving to Right	Users start from the far left edge of the 4 Cell pattern on the center line of the 4 Cell pattern and all move towards the right.

Pattern	Model Type	Description
4 Cell	Random Leaving	Users start from random locations on the horizontal center line of the 4 Cell pattern and all move towards the left.
4 Cell	Arriving to Left	Users start from the far right edge of the 4 Cell pattern on the center line of the 4 Cell pattern and all move towards the left.
4 Cell	Random	Users start from random locations on the horizontal center line of the 4 Cell pattern and move in random left or right directions.

Type in the number of users you wish to be in the model and select the kind of behavior you wish these users to have. Constant means that once a users begins its movement, if it moves outside the pattern or reaches its destination, it doesn't move. Regenerate means that the user will be regenerated with a completely new set of attributes when it moves outside the pattern boundaries or reaches its destination.

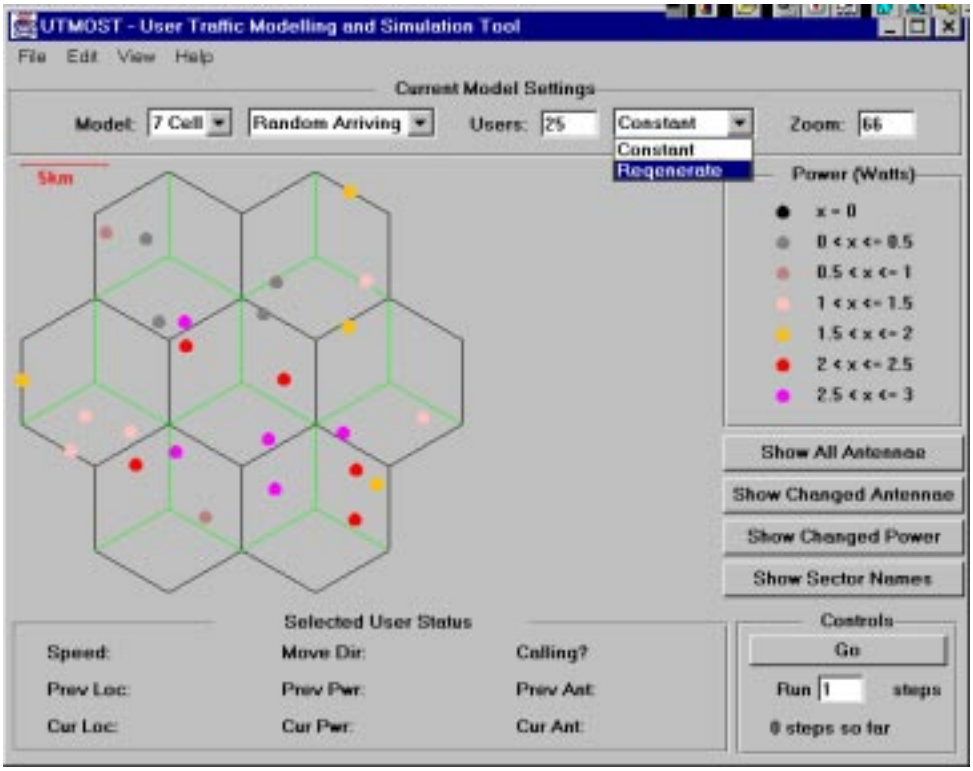


Figure 4. User Behavior Selection Pop-up Menu

Once the model has been defined, you may press "Go" to run 1 or more steps. To run more than 1 step, change the "Run steps" number. You may also wish to display any of the following things by pressing the corresponding button:

Table 2. Show/Hide Buttons in UTMOST

Button Name	Description
Show/Hide All Antennae	The antenna assigned to each user is indicated by a black line from the user to the antenna.

Show/Hide Changed Antennae	After any step in which a user changes its assigned antenna, a green line will be shown from the user back to its previous assigned antenna and a red line will be shown from the user to its newly assigned antenna.
Show/Hide Changed Power	Any user whose power has changed will be shown with a red circle around the user dot.
Show/Hide Sector Names	The sector names will be shown in each sector. A name of 5B means the cell named 5 and sector named B.

4.1.3 What Does It Do? (the files generated)

Every time "Go" is pressed in UTMOST, all users' data is updated according to their traveling directions and speeds. The current state of each user is stored in an activity data file. If a user crosses a boundary from one sector to another, then an entry for this user is made in the boundary data file. This information includes the exact time and place of boundary crossing. More information about these files may be found in Section 4.2.4.

4.1.4 Boundary Crossing Algorithms

Boundary crossing algorithms are needed to calculate exactly when and where a user crosses a sector boundary to be created. The algorithms involves tracking the sector edges. In UTMOST, the sector edges are maintained in a LineSegment array. The LineSegment class includes the two (x, y) coordinates which define the line segment in addition to the slope (m) and intercept (b) for the line. That is, m and b are the constants in the equation $mx + b = y$ which defines the line on which the two coordinates lie. The application specific addition to this LineSegment class is a two element array of integers. This array is the sectors array which will keep track of the one or two sectors which use or share this line segment.

A user is known to have crossed a boundary when its current sector location is different from its previous sector location. With the LineSegment array to keep track of all edges in the cell layout pattern, the exact boundary which the user crossed may be calculated using a LineSegment method called intersects(). This method will determine the exact point of intersection, too.

One method which will work, but requires intersects() calculations for every single element in the LineSegment array is referred to as the Brute Force method. Intersections are checked for each user movement and each LineSegment in the array. When intersections are found, the time of intersection is found and recorded. The distance from the previous location and the boundary crossing point divided by the traveling speed of the mobile user yields the boundary crossing time from the previous simulation time.

If the current sector and the previous sector are immediate neighbors, then the shared sector edge and the traveling line of the mobile user can be used to the decide the boundary crossing point. That is, only one call to intersects() will be needed. The distance from the previous location and the boundary crossing point divided by the traveling speed of the mobile user yields the boundary crossing time from the previous simulation time. However, a user may cross sector boundaries many times between two simulation steps. This method will not work.

The more optimum method uses the knowledge that a user is traveling at certain direction and must cross an adjacent sector's line segment to get to its new location. First calculate the four directions from the mobile stations to the four corners of the current sectors. Then check which two corner directions the traveling direction falls between. The selected two sector corners decide the line segment, which the user will cross the sector boundary. The intersection of the selected line segment and the traveling direction line is the boundary crossing point. The distance between the current location and boundary crossing point divided by the traveling speed yields the boundary crossing time relative to that of the last simulation step. Note that the above process will be repeated until the user reach the current sector.

Boundary crossing algorithms also need to handle the special case where the mobile user is traveling along the sector edges. Further discussion of this topic may be found in Section 4.3.

4.1.5 Perimeter Placement of Users

One of the models that UTMOST has is the "7 Cell, Edge Arriving" model. This model has all users starting from the perimeter edge of the 7 Cell pattern. To place users in this initial state, UTMOST uses the LineSegment Array to find all sector lines that only have one sector in the sector array. That is, UTMOST finds all lines for sectors which do not share with another sector. Once these edges are found, the users are randomly assigned to one of these edges. Then a random (x, y) location is found on that edge and this location is used for the user's location. Further discussion of this topic may be found in Section 4.4.

4.1.6 Custom Inside Function

An important piece of user information needed is the current sector location of the user. Every time a user moves, that user may move into a new sector (boundary crossing) or may stay in the same sector. The first approach to calculate the sector location was to use a Java built-in polygon method called inside() [11, 13, 14]. This proved to be insufficient. UTMOST also handles the special case where a user is exactly on an edge of a sector.

During the development of UTMOST, it was quickly learned that the polygon functions based in Java are for on-screen graphics and therefore, only use integer (x, y) coordinates. This was not accurate enough for calculating user locations in UTMOST. Instead, a custom calcInside() method was developed to handle floating point (x, y) and floating point sector coordinates.

Further discussion of calcInside() may be found in Section 4.2.9 and in Section 4.5.

4.1.7 Visible Sector Antennae Calculations

One of the attributes of the User class is a visible sectors string. This string is calculated just before user activity data is written out to the activity file. That is, after the user's next position is calculated, the new user information is used to determine the sector antennae which are now visible to this user. Two approaches were taken to find the visible sectors. Both of them involve figuring out the angle between two lines.

In the first approach, the law of cosines is used to calculate the angle between the user's location and each of the three lines leaving each base station. The angle made with the visible sectors lines will be less than 120 degrees.

In the second approach, knowledge about the direction in which a base station antenna points (the Sector antenna direction attribute) is used along with the direction of movement from the base station to the user's location. More discussion of the calcVisibleSectors() method may be found in Section 4.2.11 and Section 4.6.

4.2 UTMOST Design

This section discusses the design of UTMOST in detail. The classes used to describe the Users, Sectors and other things in the model are discussed as well as some of the more important methods. In addition to this section, you may wish to refer to the actual javadoc documentation found with the source. Javadoc is a Java commenting tool which provides the ability to take in-line block comments from the source and to put these comments into formatted HTML web pages. Javadoc comments are comments in the source that begin with "/**" and end with "*/" That is, there is an extra '*' at the starting block comment token. When the javadoc command is run on the *.java source files, the HTML web pages are created.

4.2.1 Coordinate System Used in UTMOST

All (x, y) coordinates are with (0, 0) located in the upper left corner of the MapArea. This facilitates drawing of objects on a GUI where (0, 0) is located in the same place.

The movement directions of a user or the direction in which an antenna points is as follows:

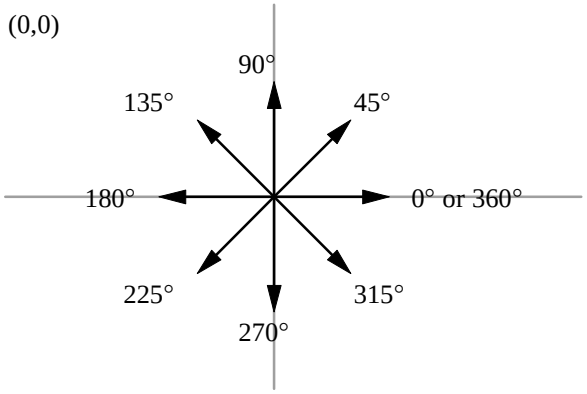


Figure 5. User Movement Directions

4.2.2 Sector Naming Schemes

A cell in UTMOST is the hexagon shape. A sector in UTMOST is one third of a cell and also represents the area covered by one of the three sector antennae. A base station is at the center of each cell. Each Sector has an external name and an internal name. The external name consists of a cell number and a sector letter (A, B, or C), e.g., 4A or 6C. The internal name is an integer number. Below, in Figure 6, is the 7 Cell and 4 Cell sector naming schemes used in UTMOST.

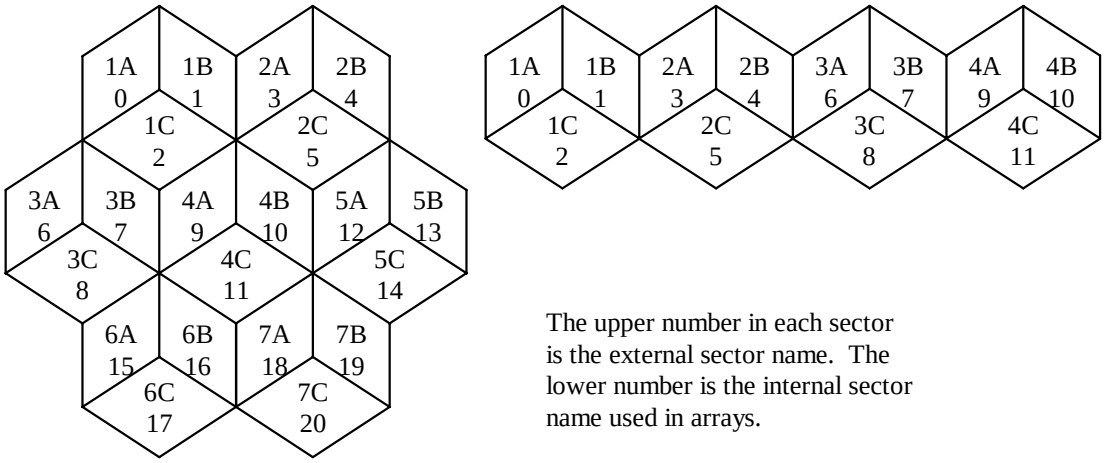


Figure 6. Sector Naming Schemes

4.2.3 All Classes Summarized

This section summarizes the more important classes used in UTMOST.

Before continuing, here are some comments about Java's Abstract Windowing Toolkit (AWT) and its building blocks [27, 28, 29, 30, 31]. A Frame in Java is the basic application window which contains pull-down menus, the title bar at the top and other traditional window features which most GUI programs have. In terms of Java class hierarchy, most objects in the Java AWT are components. A special type of component is a container; that is, the container class extends or inherits from component class. The Frame object in UTMOST is an instance of the UTMOST class. The UTMOST class has as its components, various Panels like ModelSetup, MapPanel, and UserStats. Some of these Panels have components, too. That is, some of the Panels are not only components of the UTMOST class, they are also container classes holding other Panels or Buttons or other components.

4.2.3.1 UTMOST

This is the main class in the UTMOST program. One instance of this class is created and shown. All interaction with UTMOST is through this main class and its components.

4.2.3.2 User

Each user in the model is represented by an instance of the User class. A User array of up to 5000 Users is maintained by UTMOST.

4.2.3.3 Sector

A PCS cell consists of a base station and three directional antennae. The cell is a hexagon shaped area currently hard coded to have a 5km radius. The three directional antennae break the cell up into sectors. That is, a cell is divided into three parallelograms called sectors. The Sector class represents the parallelogram. An array of Sectors is maintained for the current pattern - 7 Cell or 4 Cell. More information about Sectors may be found in Section 4.2.6.

4.2.3.4 LineSegment

To facilitate user boundary crossing locations, perimeter locations of users on the model and other things in UTMOST, the LineSegments which make up all sectors in UTMOST are stored in a LineSegment array. A LineSegment is, as its name indicates, a definition of a line segment. It has two (x, y) coordinates and some other straight line definitions maintained like the line's slope (m) and the line's y-intercept (b) as used in the mx + b = y representation of a line. More information about LineSegments may be found in Section 4.2.7.

4.2.3.5 ModelSetup

The ModelSetup class is a Java Panel which controls the model parameters. Selection of model pattern, model type, number of users, behavior of users and zoom factor is done from the ModelSetup Panel.

4.2.3.6 MapPanel

The MapPanel class is a Java Panel which controls the MapArea and it's display characteristics. It has at its left, a MapArea. It has at its right, the power legend (colors for users in the MapArea) and some buttons which allow for showing on the MapArea sector names, all assigned antennae, changed antennae and changed user power.

4.2.3.7 MapArea

The MapArea class is the Java Canvas on which the visual view of the UTMOST model is displayed. A Canvas in Java is a place where drawing may be done. The MapArea is ultimately placed in the MapPanel which is placed in UTMOST Frame for display.

4.2.3.8 UserStats

The UserStats class is a Java Panel which controls the model execution ("Go" button) and also displays selected user's current statistics.

4.2.3.9 Util

This class contains some static utility methods. This class is a Java final class which means that no instances of it may be created and methods in the class may be referred to typing, for example, Util.calcTheta(). This is much like the Java Math class which contains methods like Math.cos(), etc.

4.2.3.10 DebugLevel

UTMOST prints little or no information to the console as the default. However, at certain times during debug of the program it is helpful to display lots of debug statements. The DebugLevel class provides the switches needed to control how much and what kind of information is displayed during the execution of UTMOST.

One or more of the DEBUG variables may be passed in from the call to the Java virtual machine with the -D option, information will be printed to the Console window and to the window from which UTMOST was started. That is, to call for FLOW_DEBUGGING, start UTMOST as follows:

```
java -DDEBUG_FLOW UTMOST
```

The available DEBUG variables are listed below in Table 3.

Table 3 DebugLevel Flags

Debug Flag	Description
	The default... No debug statements will be displayed
DEBUG_TIMING	Show timing information. Currently this doesn't work
DEBUG_TRACE	Show stack traces. Currently this doesn't work
DEBUG_SECTOR	Show sector debugging information
DEBUG_FLOW	Show flow debugging information
DEBUG_MOVE	Show user movement debugging information
DEBUG_SEGMENT	Show segment calculations debugging information
DEBUG_INTERSECT	Show intersection calculation debugging information
DEBUG_USER	Show misc. user debugging information
DEBUG_VISIBLE	Show visible sector antennae calculation debugging information
DEBUG_INSIDE	Show inside sector calculation debugging information

4.2.3.11 Other Miscellaneous Classes

Below are descriptions of other classes which UTMOST uses but that are not directly related to the main UTMOST design.

4.2.3.12 Additional Panels and Dialogs

Various hooks have been placed into UTMOST to provide extensions to its current functionality. One of the hooks is a set of properties which may be changed via the "Edit → Properties..." pull down menu item. The properties that were envisioned were Variables, Files, Users, and Sectors. The Variables panel includes things like cell radius, time increment, and user dot size. The Files panel allowed changes to the activity files path, the boundary files path and the console log file. The Users panel was to allow initial input of user data from a file, instead of using the Model Setup to initialize. The Sectors panel was to allow for the customized setting up of sectors.

None of these panels work in V1.0 of UTMOST - they do display when selected, but the data from these panels is not used.

UTMOST also has some classes which are used to display "About" and "Quit" dialog boxes.

4.2.3.13 Borrowed Code (Freeware)

JDK 1.0.2 does not contain floating point formatting like is used in printf statements in C. A class called Format allows a crude but effective way of formatting floating point

numbers with various numbers of digits and significant digits. This class is from the book Core Java [30

The etched boxes around many of the main sections of the UTMOST Frame use classes from David Geary's book, Graphic Java [28]. These include DrawnRectangle, EtchBox, EtchedRectangle, and Etching classes.

4.2.4 The Files Generated

Once settings for the model are selected, the "Go" button is pressed to initiate 1 or more steps in the model. Two directories are created for each simulation run. An "activity/<date_time_stamp>" directory contains information for each user at each step of the simulation. This directory also contains the sector_data used for this particular simulation run.

Another directory called "boundary/<data_time_stamp>" contains information about any user who has crossed a sector boundary at each step of the simulation. Results from each simulation step are stored in separate files with applicable users reported in that file.

The files in both the "activity/<data_time_stamp>" and the "boundary/<data_time_stamp>" directories are named the number of the step which was executed. That is, step one's results will be in a file called "1", step two's in "2", etc. The files with name "<#> done" are used to indicate a tool has finished generating the data file and allow UTMOST and other tools such as optimization packages and simulators to work together. See Figure 7.

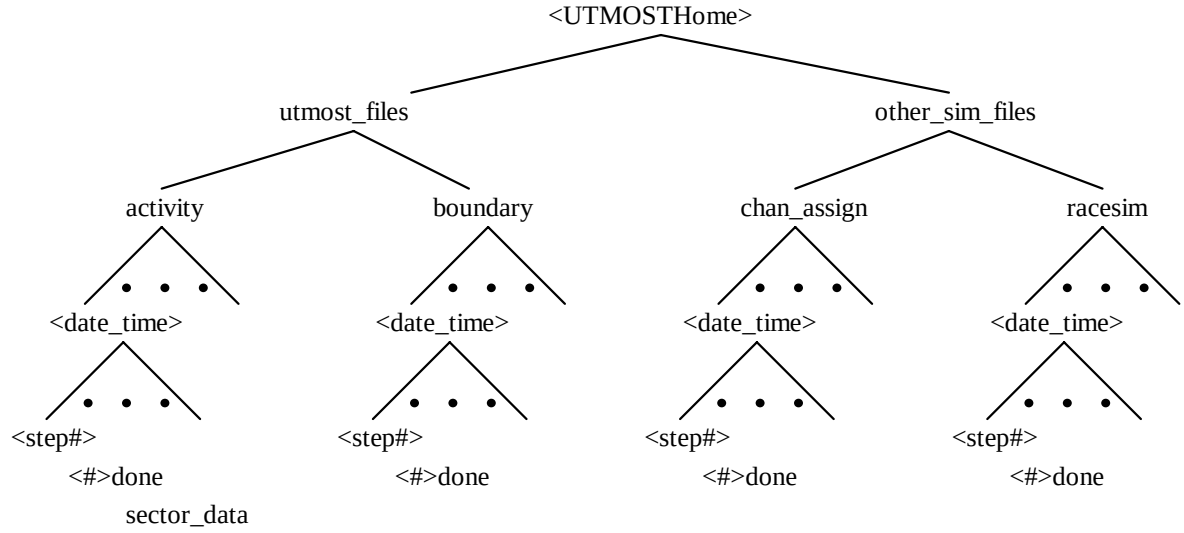


Figure 7. The UTMOST Result Files Hierarchy

<UTMOSTHome> is the directory in which the UTMOST program is found.

<data_time> uses files named YYMMDDHHMMSS of the simulation run.

other_sim_files refers to data which UTMOST may someday read from to interact dynamically with other discrete event simulators.

Detailed description of each file generated or used by UTMOST may be found in the file with the documentation called "datafiles" or in the user's guide. The "datafiles" document is in the appendix in Section 0. The user's guide is in appendix in Section 12.

4.2.5 LineSegment Class

This class is a line segment definition class. That is, the class represents a linear line segment with a starting (x, y) point and an ending (x, y) point. Based on these coordinates, the mx + b = y information is calculated (m is the slope and b is the y intercept). This class also contains a few very key methods used by the UTMOST model. Perhaps most important is the intersects() method. This method will set a User's boundary crossing (x, y) coordinate to the intersection of this line segment with a passed in line segment, if there is an intersection. If there is no intersection, the user's values boundary crossing (x,y) are not changed. This intersect method is used by UserStats' writeBoundaryResults method.

4.2.6 Sector Array

The Sector class contains information about the one third sections of a hexagon cell. The information includes an integer name, an integer corner name, a double (x, y) coordinate representing the base station location for this sector, a direction, radius and range. When an UTMOST pattern is selected, a Sector array is filled with information about all sectors in the current pattern. This is generally read out of the 7_sector_data file or the 4_sector_data file for “7 Cell” and “4 Cell”, respectively.

4.2.7 LineSegment Array

After the Sector array is loaded, this information is used to create the LineSegment Array. The LineSegment Array contains information about every line used to draw the sector patterns in the MapArea. The picture in Figure 8 shows visually how the LineSegment Array (ln[]) is created. The coordinates of the dark lined sector are stored in the p_x and p_y arrays. By looking at the index of the ln[] array, the order in which the array is built may be seen.

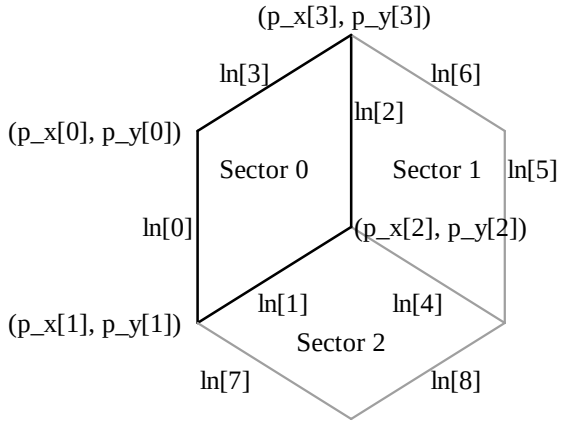


Figure 8. LineSector Array Initialization

Below is the sequence of events logically that happen to generate the ln[] array. It may help in explaining the logic.

1. Sector 0 is analyzed:
   - ln[0] is initialized and the LineSegment sector array has it’s first element set to Sector 0.

- ln[1] is initialized and the LineSegment sector array has it's first element set to Sector 0.
- Figure ln[2] is initialized and the LineSegment sector array has it's first element set to Sector 0.
- ln[3] is initialized and the LineSegment sector array has it's first element set to Sector 0.

2. Sector 1 is analyzed:
   - ln[2] is found to already exist in ln[] array and this is the first line in Sector 1. Add Sector 1 as the second element of the LineSegment sector array.
   - ln[4] is initialized and the LineSegment sector array has it's first element set to Sector 1.
   - ln[5] is initialized and the LineSegment sector array has it's first element set to Sector 1.
   - ln[6] is initialized and the LineSegment sector array has it's first element set to Sector 1.
3. Sector 2 is analyzed:
   - ln[1] is found to already exist in ln[] array and this is the first line in Sector 2. Add Sector 2 as the second element of the LineSegment sector array.
   - ln[7] is initialized and the LineSegment sector array has it's first element set to Sector 2.
   - ln[8] is initialized and the LineSegment sector array has it's first element set to Sector 2.
   - ln[4] is found to already exist in ln[] array and this is the last line in Sector 2. Add Sector 2 as the second element of the LineSegment sector array.

All Sectors in the Sectors Array are analyzed in this fashion and thus initializing the LineSegment Array.

4.2.8 calcLength() and calcTheta()

calcLength() and calcTheta() are two Util class methods which perform the geometry calculations for length and theta, respectively. The calcLength() method takes as arguments two (x, y) coordinates and returns the length of the line segment defined by these points. It uses the simple geometric equation for distance between two points [15]:

$$\sqrt{(\Delta x)^2 + (\Delta y)^2}$$

The calcTheta() method has two signatures. The first takes as arguments two (x, y) coordinates and a LineSegment. calcTheta() figures out the angle in degrees between the first point (x, y), the second point (bx, by) and the point on the line segment which is not the same as (bx, by). That is, the angle calculated is the angle at the point (bx, by). The second signature of calcTheta() takes as arguments three (x, y) coordinates and finds the angle between the first, second and third points. That is, the angle calculated is the angle at the second point.

To calculate the angle, calcTheta uses the Law of Cosines [16]:

$$\text{angle } C = \cos^{-1}\left(\frac{a^2 + b^2 - c^2}{2ab}\right)$$

where the triangle is defined as follows in Figure 9. a, b, and c are the lengths of the sides and A, B, and C are the angles shown.

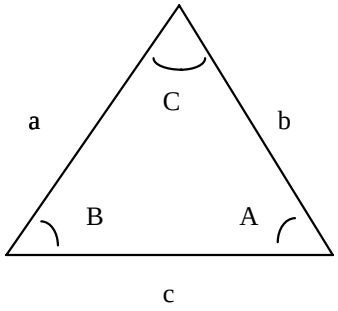


Figure 9. Triangle in Law of Cosines definition.

4.2.9 calcInside()

The MapArea class method, calcInside(), does an inside polygon check using floating point numbers instead of the integer values that the Java AWT Polygon class method inside() uses. The algorithm used in calcInside() is based on the fact that the sum of all angles inside a triangle should equal 180 degrees.

This method is described in detail in Section 4.5.

4.2.10 calcCurSector()

This method is part of the MapArea class (it is also called from the User class). To calculate in what sector a given (x, y) coordinate is located, this method creates sector polygons and uses the calcInside() method to figure out if a point is inside that sector. The calcCurSector also uses the onPerimeter() method to check the special cases where a user is located exactly on the lines or on a vertex point defining the sector polygon [15].

4.2.11 calcVisibleSectors()

This method is found in the UserStats class. This method figures out what sector antennae are visible to a given (x, y) coordinate.

This method is described in detail in Section 4.6.

4.2.12 writeBoundaryResults()

This method is in the UserStats class. This method first finds out which boundary the user crossed, where it crossed that boundary and when the cross happened. Once an intersection is found, an entry is made for this user in the boundary output file.

4.2.13 calcNextPosition()

This method is in the User class. This method will compute the next position for the user based on its attributes and the current kind of model. The model type and pattern is passed in as a parameter to calcNextPosition().

Calculations for next position use cosine and sine of an angle to determine the distance traveled. Currently, distance traveled in one time increment is hard coded at 10 second increments. Therefore, 0.0027777 times current move speed will equal the distance traveled (d). Then the new x coordinate of location will equal the old x location plus d times cosine of the move direction, and likewise for y using sine instead of cosine. Note that the above example is the condition for a user moving between 270 and 360 degrees. To handle the other quadrants of user movement, the signs are changed from both additions to appropriate addition and/or subtraction. The diagram in Figure 10 may help in understanding user movement directions.

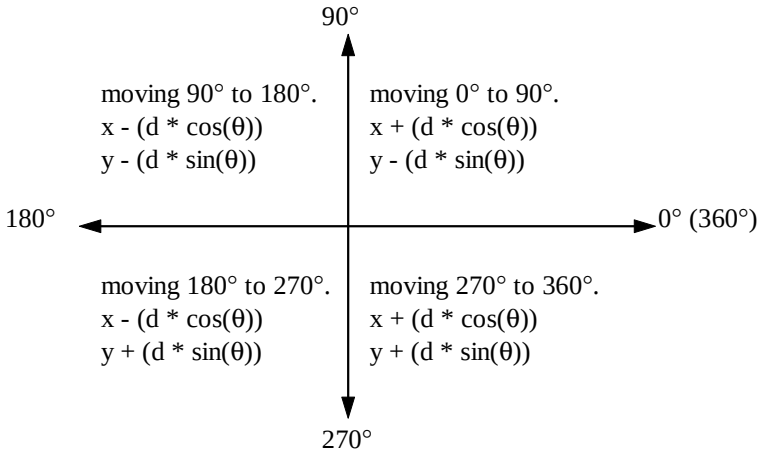


Figure 10. User Movement and Distance Moved Calculations

4.3 Boundary Crossing Calculations

As described in the introduction, the LineSegment array is used to assist in finding the exact location where a user crossed from one sector to another - the boundary crossing location. This section will discuss in more detail the two approaches used during the development of UTMOST.

As stated before, the LineSegment class is made up of two (x, y) coordinates, a slope (m), an intercept (b) and a sector array which contains the one or two sectors which use this line in their definition. When it has been determined that a user has crossed a boundary by the fact that its current sector location is different from its previous sector location, the exact location (and time) of the crossing is calculated.

This section will examine the brute force and the thought to be more optimum of finding the boundary cross locations. These methods are implemented in the writeBoundaryResults() and the writeBoundaryResultsOptimum() methods, respectively. The methods may be found in the UserStats class. The methods handle users who cross multiple boundaries in a step and will write multiple lines into the boundary cross files if a user crosses multiple boundaries in a step. The brute force does not guarantee order in the file, but the optimum method will be in order. The optimum method will display a user's first boundary crossing first, second crossing second, etc.

Both methods use the LineSegment intersects() method. This method solves the two simultaneous linear equations to get the intersection of the two lines. Then the

intersection point is checked to see if it is within the bounds of both line segments. Then and only then the segments are deemed to intersect.

4.3.1 Brute Force Calculations

The brute force method creates a LineSegment for the previous to current step's user movement. Then the algorithm traverses the entire LineSegment array and checks to see if the user movement crossed any of the line segments. If a cross is found the results are written to the boundary cross file. The entire line segment array is traversed, so that any line that has been crossed will be found and recorded.

The pseudo code for this method is as follows:

```
for each line in LineSegment array {
  if (userMove intersects line) {
        writeBoundaryEntry
        }
}
```

4.3.2 More Optimum Calculations

In the more optimum algorithm, the LineSegment array is used to find adjacent sectors to the previous sector. The line segments which makes up the current line segment are checked in order. The checks are stopped as soon as the intersecting line segment has been found. Once the line segment which the user's travel intersects is found, the new adjacent sector variable is set to the adjacent sector to the previous sector. This information is written to the boundary file.

Now the previous sector is set to the new adjacent sector so the loop may begin again. The loop only needs to be performed again if the adjacent sector is NOT equal to the current sector. That is, if the user has crossed more than one boundary the check must be performed again.

When UTMOST must check for a second line segment crossing, it must NOT find the segment it just found. To do this, a simple "crossedLines" array is used to flag any line which has already been crossed. Therefore, for any check, the crossedLines array must first be check to make sure this is a valid line at which to look.

This segment searching loop continues until the adjacent sector found does indeed match the current sector location of the user.

The pseudo code for this method is as follows:

```
prevSector is where we start
adjSector is the sector we find we have crossed into
curSector is the ultimate location of the user.
prevSector becomes adjSector before each loop of the while
loop.
while not done (done when adjSector is equal to curSector)
  for each line in LineSegment array {
        if this line has not been crossed before
              if this line shares an edge with prevSector
```

```
                        if (userMove intersects line)
                              update crossedLine array
                              update adjSector to be sector we
                                    crossed into.
                              writeBoundaryEntry (uses both
                                    prevSector and adjSector)
                              update prevSector = adjSector
                        }
                  }
            }
      }
}
```

4.3.3 Timing Comparison of the Algorithms

Ironically, timing tests of the brute force method and the more optimum method show that the brute force method actually works faster!!!

In order to obtain some timing numbers, a debug flag was added to UTMOST which, when encountered, would print a message and a time to the console. The flag was set to print a "started" time at the beginning of the main step for loop in the goButtonAction() method. It was set to print a "completed" at the end of this loop. That is, the timing includes the file I/O and various other system overhead in addition to the updating of the user's boundary cross data.

Data was collected for the following conditions:

- Optimum (both the improved boundary cross and visible sector algorithms incorporated)
- Visible Sectors Optimization Only
- Boundary Sector Optimization Only
- No Optimizations (both the old boundary cross and visible sector algorithms were used)

Five runs were completed for each condition above. Each run used 100 Users, Random movement pattern, Regenerating User behavior and 20 steps per Run. In addition to the random speed assignments given by UTMOST, UTMOST was modified to allow a second set of data to be collected which had each user moving at a very fast 600 km/hour.

The summary results are as follows where the percentages reflect how much faster (or slower if negative!) the method is compared to the no optimization numbers.

Table 4. Users Moving at random speeds (slow)

Code Timed	Total Time	Avg per Cross Calculation
Optimum	11%	1%
Visual Sectors	9%	8%
Boundary Cross	-3%	-9%
No Optimizations	0%	0%

Table 5. Users Moving at 600 kph

Code Timed	Total Time	Avg per Cross Calculation
Optimum	7%	10%
Visual Sectors	5%	6%
Boundary Cross	-3%	-3%
No Optimizations	0%	0%

Since these numbers seem surprising with the Boundary Crossing optimization showing slower performance than the original UTMOST code, some more calculations on the data were performed. The number of boundary crossings at each step were totaled and a number representing the average time taken per crossing was calculated. Then the comparisons were made between using old code and optimized code. Those results may be found in Table 4 and Table 5. These numbers to do not show much more except that the Optimum average time per crossing is insignificantly better when few boundary crossing calculation are needed. That is, the 1% number for average per crossing calculation on the random speed model shows that when users don't cross boundaries often, the algorithms don't have much effect!

The thought-to-be-optimum method requires much bookkeeping in order to implement. There are only fifty one (51) line segments in the 7 Cell pattern. The overhead of the thought-to-be-optimum method is more than the benefit of only performing four intersect method calls. This method, however, may become faster than the brute force method if many more cells are in the model's pattern.

4.4 Perimeter Placement of Users (Initializing)

As described in the introduction, the model "7 Cell, Edge Arriving" requires the initial placement of users to be on the perimeter edges of the 7 Cell pattern. This section will discuss this algorithm in more detail.

The LineSegment array of sector lines is used to find the perimeter edges. An edge line will only have one entry in its sectors array. That is element 0 of the sector will be the only sector which uses the edge and element 1 will still contain its initialized value of -1. A -1 means that this is not a sector. All sectors have integer names equal to or greater

than 0. To find all the edge line segments, the LineSegment array is searched and a new array of edgeLines is created.

Once this edgeLines array is filled, users are randomly assigned to an edgeLine. Then using some LineSegment methods, a random point on the line is selected for this user's initial perimeter location. The methods in LineSegment are: getRandomXPoint() and calcYPoint() for all lines that are not vertical and getX1() and getRandomYPoint() for vertical lines. The methods do pretty much what their names indicate. getRandomXPoint() and getRandomYPoint() will do just that: they will return a random x or a random y point which falls on the line segment. calcYPoint() takes a single argument of x. Given x, the line segment definition of m and b are used to calculate the corresponding y value on this line segment. That is, y is calculated to be mx + b. getX1() is a method which returns the value of x1 where x1 is an attribute of the line segment. When a line is vertical, x1 will equal x2. In other words, all values of x will be the same.

4.5 Calculating User's Sector - calcInside().

As described in the introduction, an important piece of user information needed in UTMOST is the current sector location of a user. During the development of UTMOST, Java's Polygon class [11, 13, 14] and its inside() method were discovered to be inadequate. In UTMOST, a custom calcInside() method was used instead of the Polygon class and its inside() method.

4.5.1 Problem with JDK AWT Built-in Functions

The Java Polygon class consists of two arrays of coordinates and an integer variable indicating how many points there are. See the Polygon constructor below:

```
public Polygon(int xpoints[],int ypoints[],int npoints)


Constructs and initializes a Polygon from the specified
parameters.
Parameters:
xpoints - the array of x coordinates
ypoints - the array of y coordinates
npoints - the total number of points in the Polygon
```

The two arrays are for the x and y coordinates respectively. That is, the 0 index of the x array is paired with the 0 index of the y array. The arrays are arrays of integers! This is the main problem with the Polygon class.

Users in UTMOST are represented by an (x, y) coordinate. The coordinate is made up of floating point numbers. Sectors in UTMOST are defined by an (x, y) point representing the base station for that sector, a direction at which the antenna for this sector points, a radius of the cell of which this sector is a part, and the correct offsets from the base station to create the sector. All of the points, radius and offsets are floating point values, also.

In trying to use the built-in Java Polygon class, the coordinates are converted to integer values. The conversion introduced error which caused some users boundary cross

to be discovered at the wrong execution (Go) step of the model. Boundary crossings from one sector to another must be found in the modeling step in which they happen. This is why the calcInside() method needed to be developed.

4.5.2 UTMOST Implementation

The MapArea class method, calcInside(), does an inside polygon check using floating point numbers instead of the integer values that the Java AWT Polygon class method inside() uses. The algorithm used in calcInside() is based on the fact that the sum of all angles inside a triangle should equal 180 degrees.

First a sector is divided into two triangles. The first triangle uses the first three points in the polygon defining the sector. That is, the 0th, 1st and 2nd points of the polygon definition array are used. The second triangle is the 2nd, 3rd and 4th points of the polygon. Then, using the same algorithm on each triangle, all of the inside angles are calculated as shown in Figure 11. If point (x, y) is inside the triangle, the sum of A through F angles will be 180 degrees.

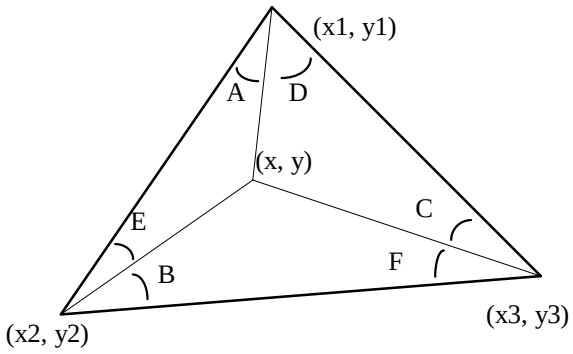


Figure 11. Triangle Angles for calcInside() Method.

If (x, y) is outside the triangle, the sum of A through F will be greater than 180 degrees. A point outside the triangle would look like Figure 12.

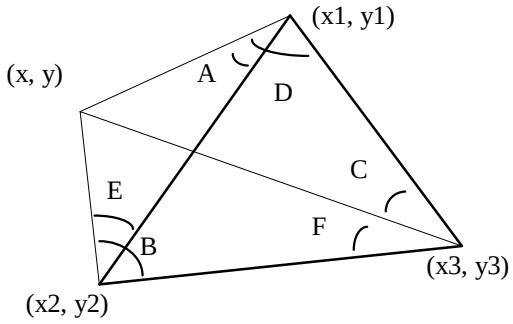


Figure 12. Triangle with (x, y) Outside the Triangle.

4.6 Determining Visible Sector Antennae

There were two ways that this method was implemented. One uses no knowledge about the sectors antenna direction and performs three calcTheta() calls. The second only does not use calcTheta but uses the Util.getDirection() method and uses the knowledge about a sector's antenna direction (Sector.getDirection() method). The calcVisibleSectors() and calcVisibleSectorsOptimum() methods may be found in the UserStats class.

4.6.1 Method Which Uses Three calcTheta() Calls

The first is done by calculating visible sector antennae to a user by using the base station coordinates and the sector lines which share the base station locations. That is, given a base station (bx, by) coordinate, there are three line segments which project out from this point. Then, using the line from the user coordinate (x, y) and the base station (bx, by) and each of the three base station line segments, the angle made is calculated. The sector antenna which is visible to the user is the sector antenna who owns the two lines which have an angle with the point (x, y) which is less than 120 degrees. See Figure 13.

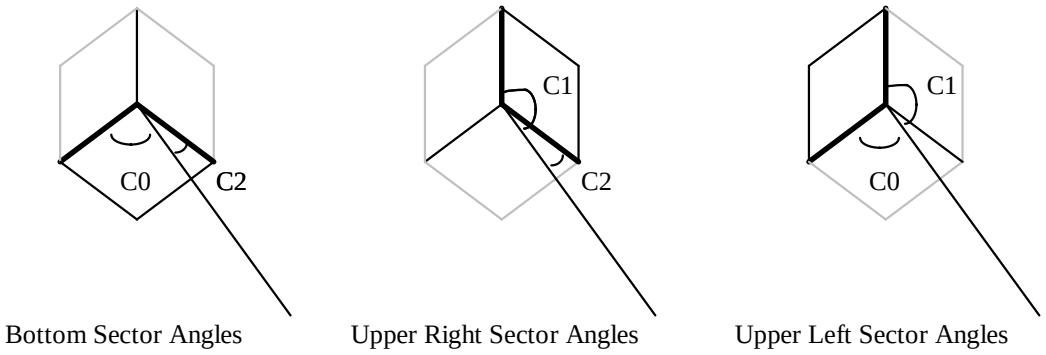


Figure 13. Visible Sector Antennae Angles.

4.6.2 Method Which Uses One calcTheta() Call

The second method for calculating visible sector antennae uses the knowledge of the antenna direction and then the direction that this user is in relation to the base station. This method depends on the sector antenna's direction and the angle from the base station to the point (x,y). It does not use the Law of Cosines.

A sector's direction attribute indicates the center of the 120 degree range of the sector antenna. This means the range visible by the sector antenna is from (direction - 60 degrees) to (direction + 60). Calculating the direction from base station to (x, y) can be done by just calling the user Util.getDirection() method.

4.6.3 Timing Comparison of the Algorithms

Summary timing results are displayed in Table 4 and Table 5 in Section 4.3.3.

There does seem to be a significant (albeit small) improvement in performance using the optimized algorithm. The data was collected in the same manner and at the same time as the timing date for the boundary cross algorithms. The descriptions in Section 4.3.3 will not be repeated here.

Anytime many sine, cosine, or other multiplication or division calculations can be removed from a piece of code, then there should measurable be performance improvements!

4.7 Lessons Learned in the Design of UTMOST

This was the first large GUI application the author designed and implemented. The task has been most rewarding and interesting. The author feels much more comfortable with the design and implementation of fairly complex user interfaces. The author gained very broad knowledge of the Java programming language and became much more proficient in object oriented design and implementation.

4.7.1 Floating Point Equivalence

Perhaps the most difficult and embarrassing lesson learned is that computers have trouble equating one floating point number to another. That is, saying that a double variable x is equivalent to 180 is not as easy as coding (x = = 180). Instead, a Util.isEquiv() method needed to be implemented which would return true if the first passed in floating point number was really, really close to the second passed in number. This fixed many boundary case errors, which were in the UTMOST code. The boundary case code was in UTMOST, but almost never got called because of the inaccuracy of "= =" with floating point numbers.

4.7.2 Graphics vs. Modeling, or Integer vs. Floating Point

Graphics packages like the one that the JDK AWT provides may be based on integer coordinate systems [32]. This is certainly good enough for displaying objects on a video monitor since there are finite pixels in the screen. UTMOST is a modeling tools for physical user movements and physical placements of sectors, cells and antennae. This requires a floating point number coordinate system. This put limitations on how the author could use the built-in AWT methods for manipulating and calculating user movements, boundary crossings, and sector locations. In particular the calcInside() method was developed because the class Polygon's inside() method was based on integer coordinates.

4.7.3 Linear Algebra Implementations

Much of the user movement, the sector placement and various other calculations used in UTMOST was based on linear relationships and geometric theorems. In particular, user movements were modeled as linear motion in UTMOST. This means a computer implementation of mx + b = y needed to be generated in addition to figuring out distances traveled in a unit of time along a given direction. Also, computing the relationship between a mobile user and a base station requires implementing theta calculations based on the Law of Cosines and using results to calculate visible sector antennae or current sector location of a user.

4.7.4 Java

Java has turned out to be a very powerful language even though it is still suffering from some immaturity in its design. The entire UTMOST program has been written in

Java. Java had enough class library features to allow creation of a fairly complex GUI and all the underlying mathematical calculations. Because of its infancy, printing current screen was omitted from UTMOST. The JDK 1.1 code is commented out in the UTMOST class which would enable the "Print..." capabilities.

Also, the way that the panels communicate with one another was probably not done in the most object oriented design way - all panels maintained a reference to the UTMOST Frame object. This top level UTMOST class is where the user array, the sector array, the line segment array and various other variables reside. With the JDK 1.1, a totally new event handling mechanism is implemented where any panel may monitor events which may originate outside its bounds. That is, if a button was pushed in the UserStats panel, the MapArea may be able to react to that event. This new event handling is done largely via EventListeners. According to a Java instructor, rather than put up with shortcomings of the old event handling (JDK 1.0.2 and older), the designers of Java just rewrote the entire thing [34].

4.7.4.1 Implementation Performed on Various Platforms

Development of UTMOST was done primarily on two machines. One was a Pentium PC running Microsoft's Windows 95, the other was a SPARC running Sun Microsystem's Solaris 2.5.1 and 2.6 (beta version). Often, bytecode class files were ftp-ed from one platform to the other and no re-compiles were performed. The class files executed just fine when moved from one machine to the other. This is the much publicized power of Java.

4.7.4.2 Not Yet Totally Platform Independent (Java Beans?)

Moving compiled byte code from one machine to another usually worked just fine, however there are still some subtle differences which may be noticed. For example, the Power Legend which appears on the MapPanel displays the colored dots at differently on a PC than on a SPARC. Also the behavior (in JDK 1.0.2) of Dialogs in PC's sometimes work different on SPARC's. When a dialog is displayed, control is not always returned to the calling program when a "Cancel" or "OK" button is pressed.

4.8 Future Directions and Thoughts on Enhancements of UTMOST

This section discusses various areas where UTMOST may be extended, improved or enhanced.

4.8.1 Extending the Number of Cells

There are two dimensions to extending the number of cells. One is orientation and one is number of cells. First, this section discusses what may be done to handle orientation. Second, this section discusses number of cells extensibility.

Currently sectors are placed by knowing a base station location for the sector and knowing that sector's internal number. This number is divided by mod 3 to figure out which direction the sector needs to face. In order to face sectors in different orientations, the direction attribute of the sector may be used instead to place sectors.

The number of cells is not restricted by the sector file definition, but instead it is restricted by the 4 cell and 7 cell switches found throughout the code. At one time during

the development of UTMOST, a 5 cell linear pattern datafile was created. The code was changed to read this file instead of the 4_sector_data file. The 5 cells were drawn correctly with the existing code. There were some problems executing the user traffic modeling steps. These should not be too hard to fix.

4.8.2 Providing Overlays of Wireless Subnetworks (Hexagons in Hexagons)

Based on the observations in the previous section and based on the another piece of information which is used to draw sectors, handling wireless overlays on top of or in place of the current 5km sized cells may be possible. The other piece of information is that sectors are drawn with only knowledge of the base station as mentioned in the previous section and a radius of the cell. This radius parameter may be used to adjust the sizes of sectors drawn. The only problem with the overlay of smaller cells is that sufficient overlap with the larger adjacent cells will be required since hexagon patterns do not subdivide like squares or triangles.

4.8.3 Interfacing to POCAT and RACESIM (Files to Sockets)

All of the interfacing and updating of user data based on POCAT or RACESIM discrete simulation tools is done via files. Empty files called <step#>done are used to signify to all of the tools that a step has been completed and results may be read. The location of this interfacing code in UTMOST is at the end of the goButton() method in the UserStats class.

In order to move towards a socket based model, this file interface code would be replaced with socket code. There would need to be a protocol to clearly define the data sent and received on the sockets. Instead of waiting for an entire step to be completed, it may be possible to perform simulation that communicate the results one user at a time.

4.8.4 Overlay on GIS Maps

Java has very good graphical features which would allow insertion of GIS maps in the background. The maps would most likely be read from one of the many GIS databases. Specific road and geological features could be read. One source of information is the U.S. Geological Survey's National Cooperative Geologic Mapping Program [35]. Another is the TIGER/Line (TM) Files from the U.S. Department of Commerce [36].

4.8.5 Customizing User Travel Patterns

Since a user's travel is calculated each step, the direction in which a user is moving may be changed each step. This would still result in linear movement from one step to another, but given small enough time increments for each step, virtually any travel pattern may be simulated.

4.9 Power Control and Channel Assignment

This section discusses the design and implementation of POCAT, a POwer control and Channel Assignment Tool, the integration of POCAT and UTMOST to RACEWIN, and the important design issues encountered.

POCAT uses the network traffic data generated by UTMOST and generates power and channel assignment data, which can be verified in UTMOST. POCAT generates the data by using an algorithm for combined cell-site selection and power control. The algorithm used is a modified version of two algorithms, one by Hanly [38] and the other by Yates and Huang [39]. The modifications include changing the network model to consider the overlapping of sector antennas angles and the soft handoff.

The prohibiting factor for frequency reuse in wireless systems is the interference caused by the environment or by other mobiles. Interference can be reduced by making use of power control and channel assignment techniques. The main idea with power control is to decrease effects of the **near-far problem**[1] [40], caused by having users transmitting on the same frequency band from different locations. By using power control, all user signals are received with roughly the same power level at the base station.

4.9.1 Network model used by channel assignment algorithms

The network model used in the algorithms in [38] and [39] can be summarized as follows :

$\boldsymbol{M}$ (1,2,..$\boldsymbol{M}$) users,
$\boldsymbol{K}$ (1,2,..$\boldsymbol{K}$) cells (base stations).
$\boldsymbol{R_k}$ is the sum of the received power from all users and noise at base station $\boldsymbol{k}$,
$\boldsymbol{\eta_k}$ is the noise measured at base station $\boldsymbol{k}$,
$\boldsymbol{h_{ik}}$ is the path gain between the $\boldsymbol{i}$th user and the $\boldsymbol{k}$th base station.

If user $\boldsymbol{i}$ transmits with power $\boldsymbol{p_i}$, then the power received at base station $\boldsymbol{k}$ is given by $\boldsymbol{h_{ik}}$ multiplied by $\boldsymbol{p_i}$. $\boldsymbol{h}$ is a snapshot of the network, since users are in practice mobile and is calculated by using an appropriate formula for path loss. To simplify the calculations, a simple formula like distance^{-4} can be used. Different formulas for path loss calculations are described later on in this paper.

At any given time, user $\boldsymbol{i}$ is served by one base station k, with an assigned power $\boldsymbol{p_i}$. The vector $\boldsymbol{c}$ represents the allocation of users to base stations, and the goal is to determine the optimal $\boldsymbol{c}$ and power vector $\boldsymbol{p}$ for any given configuration of users.

Yates and Huang in [39] does not discuss in any detail on the selection of the initial power vector. An initial power of 0 is used as an example.

4.9.2 The Minimum Power Assignment Algorithm

In [39] Yates and Huang presents an algorithm, Minimum Power Assignment (MPA), that reduces the total interference by minimizing the users transmitting power. The MPA algorithm can be summarized as follows:

1. Select the initial power $\boldsymbol{p_i}$ for each user $\boldsymbol{i}$.
2. Calculate the path gain (loss) $\boldsymbol{h}$ between each user and each base station.

[1] The received power from the users varies considerably when users are transmitting using the same power level from different locations. Strong signal from nearby users will interfere with weaker signal from users far away.

3. Determine the target carrier-to-interference ratio (CIR) γ for each user.
4. Determine the noise associated with each base station and calculate the vector ***R***.
5. Calculate the transmitted power at time ***t*** for each user using Equation (1-1).
6. Repeat step 4 and 5 until the power vector converges to a solution.

$$p_i(t+1) = \min \frac{\gamma_i (R_k(t) - h_{ik} p_i(t))}{h_{ik}}, k \in K$$

Equation 4-1

Step 1, 2 and 3 will not change as the power vector changes, so the only steps in the algorithm that need to be recalculated for each iteration are Steps 4 and 5.

4.9.3 The Hanly Algorithm

Hanly's algorithm [38] uses the same technique as in the MPA. One difference between the two algorithms is that Hanly's allows the possibility to have limitations in the set of available base stations for each user. That is, ***c*** is restricted by the geometry of the network, thus for each user ***I,*** there is a set D_i consisting of precisely those base stations that user ***i*** is allowed to connect to. Hanly's algorithm can be summarized as follows:

1. Select the initial power p_i for each user ***i***.
2. Calculate the path gain (loss) ***h*** between each user and each base station.
3. Determine the set ***D*** for each user.
4. Determine α_i ,which is the required bandwidth of user ***i*** and ***W*** which is the channel bandwidth (in Hz).
5. Determine the noise associated with each base station and calculate ***R***.
6. Calculate the transmitted power at time ***t*** for each user using Equation (1.4.1).
7. Repeat Steps 5 and 6 until the power vector converges to a solution.

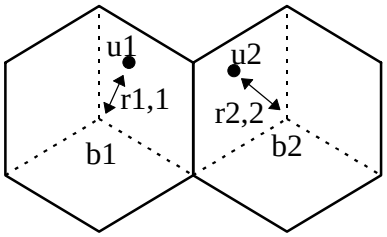
$$p_i(t+1) = \min \frac{(\alpha_i / W) * (R_k(t) - h_{ik} p_i(t))}{h_{ik}}, k \in D_i$$

Equation 4-2

Steps 1, 2, 3 and 4 will not change as the power vector changes, so the only steps in the algorithm that need to be recalculated for each iteration are Steps 5 and 6.

For comparison purpose, the notation used here to describe this algorithm is different from that used by Hanly in [38].

In most cases, the users are assigned to the sector antennae, which is closet to them. But when the traffic in a sector increases, some users will be assigned to a nearby sector. The problem is to chose the number of sectors in the set ***D,*** so that users has a sufficient number of base stations to chose from. Having too many sectors in ***D*** will create a lot of unnecessary calculation of powers level at the base stations, that are so far away that the user won't be assigned to them anyway.



Distance, r :

	Base Station1 (b1)	Base Station2 (b2)
User1 (u1)	130 m	350 m
User2 (u2)	290 m	150 m

Pathgain, h: Using r^{-4}

	Base Station1 (b1)	Base Station2 (b2)
User1 (u1)	3.50e-9	6.66e-11
User2 (u2)	1.41e-10	1.98e-9

R(1): Using 0.1W as initial power

Base Station1 (R1)	Base Station2 (R2)
3.64e-10	2.05e-10

P(1):

	Base Station1 (b1)	Base Station2 (b2)
User1 (u1)	**0.0040W**	2.97 W
User2 (u2)	2.48 W	**0. 0034W**

R(2):

Base Station1 (R1)	Base Station2 (R2)
1.45e-11	7.00e-12

P(2):

	Base Station1 (b1)	Base Station2 (b2)
User1 (u1)	**1.37e-4 W**	0.101 W
User2 (u2)	0.099 W	**1.35e-4 W**

R(3):

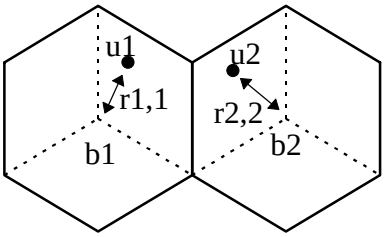
Base Station1 (R1)	Base Station2 (R2)
4.99e-13	2.76e-13

P(3):

	Base Station1 (b1)	Base Station2 (b2)
User1 (u1)	**5.44e-6 W**	0.0040 W
User2 (u2)	0.0034 W	**4.61e-6 W**

This example shows three iterations of Hanly's algorithm. After three iterations, User 1 is assigned to Base station 1, p_1 = **5.44e-6** W, and User 2 is assigned to Base Station 2, p_2 =**4.61e-6** W.

Figure 14. Iterations of Hanly's Algorithm



Distance, r :

	Base Station1 (b1)	Base Station2 (b2)
User1 (u1)	130 m	350 m
User2 (u2)	290 m	150 m

Pathgain, h: Using r^{-4}

	Base Station1 (b1)	Base Station2 (b2)
User1 (u1)	3.50e-9	6.66e-11
User2 (u2)	1.41e-10	1.98e-9

R(1): Using 0.1W as initial power

Base Station1 (R1)	Base Station2 (R2)
3.64e-10	2.05e-10

P(1):

	Base Station1 (b1)	Base Station2 (b2)
User1 (u1)	**0.0040W**	2.97 W
User2 (u2)	2.48 W	**0. 0034W**

R(2):

Base Station1 (R1)	Base Station2 (R2)
1.45e-11	7.00e-12

P(2):

	Base Station1 (b1)	Base Station2 (b2)
User1 (u1)	**1.37e-4 W**	0.101 W
User2 (u2)	0.099 W	**1.35e-4 W**

R(3):

Base Station1 (R1)	Base Station2 (R2)
4.99e-13	2.76e-13

P(3):

	Base Station1 (b1)	Base Station2 (b2)
User1 (u1)	**5.44e-6 W**	0.0040 W
User2 (u2)	0.0034 W	**4.61e-6 W**

This example shows three iterations of Hanly's algorithm. After three iterations, User 1 is assigned to Base station 1, p_1 = **5.44e-6** W, and User 2 is assigned to Base Station 2, p_2 =**4.61e-6** W.

Figure 14 shows three iterations of Hanly's algorithm. Assume a noise free environment with W=1.25 MHz, α=30kHz and a cell radius of 200 meters.

4.10 POCAT

POCAT is a Power control and Channel Assignment Tool that implements an extension of the technique used for channel and power assignment in [38] and [39]. The model used is extended to include the effect of sector antennas overlapping and the soft handoff.

4.10.1 The wireless network model

The network model used in POCAT is an extended version of the model used by Hanly and Yates. This model is using the CDMA technology as in the IS-95 North American standard. By using the CDMA technology, the frequency bandwidth can be reused in every cell, compared to older technologies where the frequencies are reused in seven cell patterns.

The network model has ***M*** (1,2,..***M***) users, ***K*** (1,2,..***K***) hexagonal cells (base stations) and **S**(1,2,..***S***) sector antennae. Each hexagonal cell consists of three sectors where each sector antenna is responsible for a 120 degree coverage area of the cell. At any given time, user ***i*** is served by at least one sector antenna, $\mathbf{s_i}$, with an assigned power $\boldsymbol{p_i}$. The use of soft handoff [47] allows the user to be served by more than one antenna at the same time during handoff to avoid the interruption.

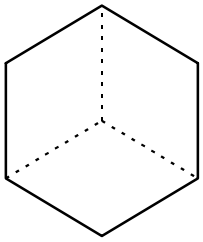


Figure 15. A hexagonal cell is divided into three sectors

We wish to determine the optimal power vector ***p*** for any given configuration of users by assigning the user to the optimal sector.

4.10.2 Visible Users

Since our model is using base stations with three sector antennae, the number of interfering users is reduced to only the users that are "visible." See Figure 16. In [38] and [39] omni-directional antennas are considered, and all users cause interference at all of the base stations.

Visible means that a user is located within the coverage area of the sector antenna as shown in Figure 16 below. The total coverage area of a sector is 120 degrees plus an overlap area. $\boldsymbol{S_i}$ is a vector with the visible sector antennas for user ***i***.

4.10.3 Sector Overlapping

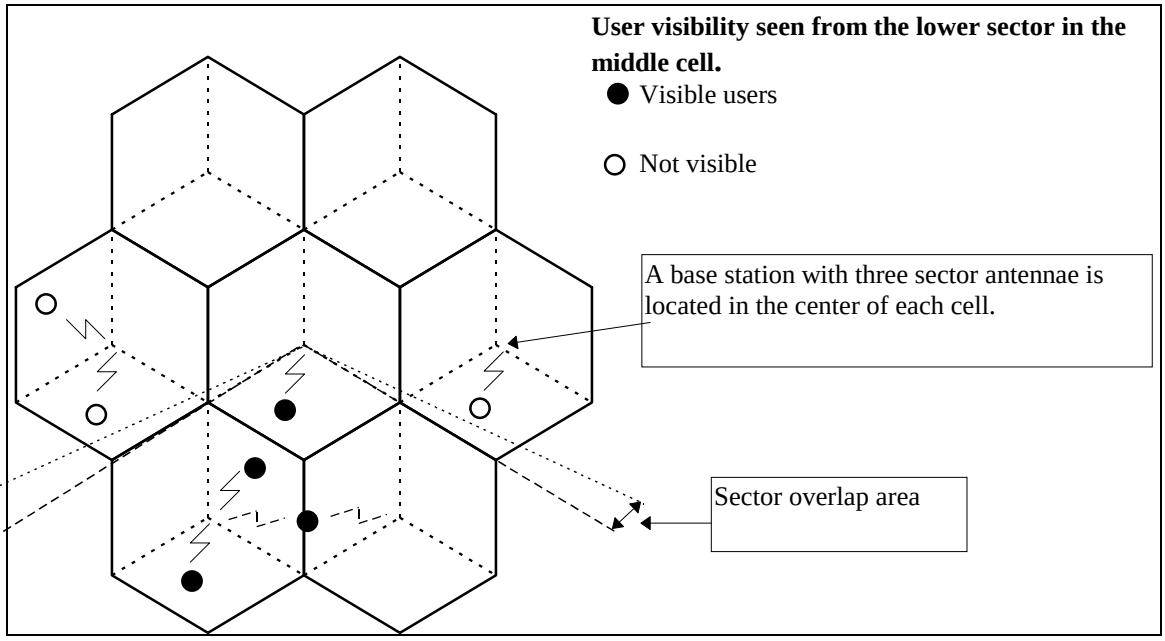


Figure 16. Visible Users

The coverage area of the antennae are not completely isolated to the 120 degrees. This means that the coverage area of the sectors are overlapping considerably which is useful for soft handoff. In POCAT, the overlap is decided by an overlap factor, which is default set to 15%. This gives us a sector coverage area of 156 degrees.

4.10.4 Soft handoff

In CDMA, soft handoffs are used to reduce the number of dropped calls and to eliminate the break in transmission during the handoff. Soft handoff allows the user to be served by two or more sector antennae in the same or different cells simultaneously during the handoff transition. A user may be in soft handoff during the entire call, if the user remains within the overlapping areas among sectors.

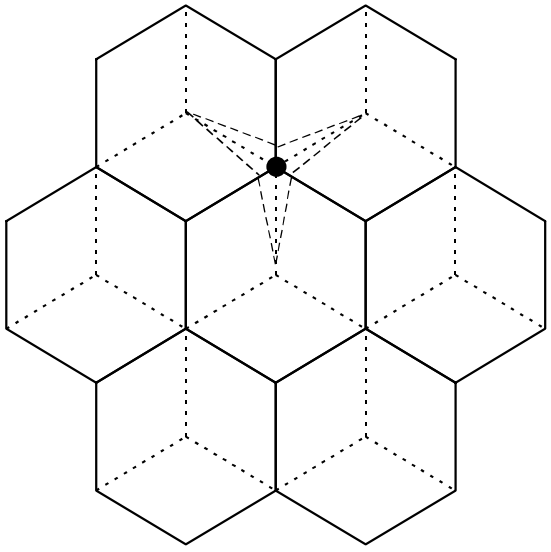


Figure 17: Soft Handoff

A user may be in a soft handoff with up to six sector antennae as shown in Figure 17. In this example, the user is at the same distance from all six base stations. The user will eventually be served by one of these antennae depending on the traveling direction. In POCAT, a user is in handoff with another sector if the difference between the two path gain values are within a specific handoff threshold. If a user is visible from two sector antennae in the same cell, the user is always said to be in handoff (softer handoff).

The following equation is used to determine if the user is in handoff or not:

$$d = \frac{pathgain_A - pathgain_B}{pathgain_A} \leq handoff\_Threshold$$

Equation 4-1

where A is the sector which the users is first assigned to, and B is one of the sectors that the user might be in handoff with. If *d* is less than zero, the user is closer to B than to A. In this case, the user will still be assigned to A and also be in handoff with B.

4.10.5 Sector Naming Schemes

Each Sector has an external name and an internal name. The external name consists of a cell number and a sector letter, e.g. 1A or 2C. The internal name is an integer number used in arrays. Below, in Figure 18, is the 7 Cell and 4 Cell network models used by UTMOST with the names of sectors.

These are just examples of the network models that are available in UTMOST currently. POCAT can handle any kind of hexagonal cell configurations, with each cell having three sectors as long as this naming scheme is used.

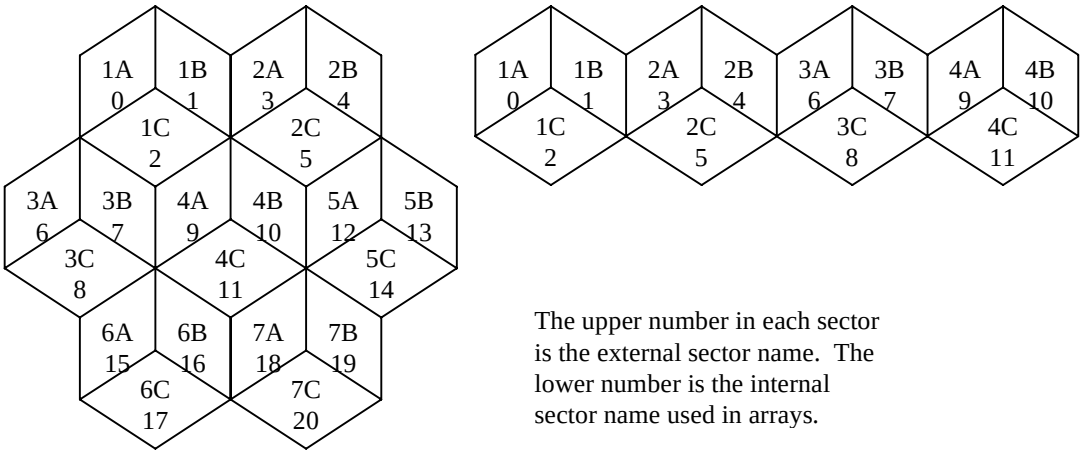


Figure 18. Sector Naming Schemes

4.10.6 Modeling Signal Fading

POCAT can use several different models for signal fading in the network. Some of the implemented models are simplified models for estimation of signal fading. The Hata model [44] is based on graphical path loss information provided from measurements by Okumura.

4.10.6.1 Single step fading law:

The single step fading law is used for modeling signal fading. The Signal fading is calculated using r^{-n}, where ***r*** is the distance from a transmitter to a receiver in meters and ***n*** is a constant between 2-4.

4.10.6.2 Multiple step fading law

Signal fading is calculated using r^{-n}, where *n*=2 for "short" distances and *n*=3 or *n*=4 for longer distances.

4.10.6.3 Friis equations for free space propagation

Friis free space equation [42] is given by

$$P_r = \frac{P_t G_t G_r \lambda^2}{(4\pi)^2 r^2 L}$$

Equation 4-2

where P_r is the received power, G_t is the transmitter gain, G_r is the receiver gain, λ is the wavelength of the transmitted signal, and L are system losses not related to propagation.

4.10.6.4 The Hata Model [44]

The Hata model is an empirical formula for path loss in an urban environment. The standard formula for urban area pass loss is given by

$$L_{50} = 69.55 + 26.16 * \log_{10} f - 13.82 * \log_{10} h_{te} - a(h_{re}) + (44.9 - 6.55 * \log_{10} h_{te}) * \log_{10} d$$

Equation 4-3

where the loss in dB, *f* is the frequency between 150 to 1500 MHz, h_{te} is the effective transmitter (base station) antenna height ranging from 30 to 200 meters, h_{re} is the effective receiver (mobile) antenna height ranging from 1 to 10 meters, *d* is the distance between the mobile and the base station in kilometers, and $a(h_{re})$ is a correction factor for the mobile antenna height and is computed as follows:

For a small to medium size city:

$$a(h_{re}) = (1.1 * \log_{10} f - 0.7) * h_{re} - (1.56 * \log_{10} f - 0.8) dB$$

Equation 4-4

For a large city:

$$a(h_{re}) = 8.29 * [\log_{10}(1.54 * h_{re})]^2 - 1.1 dB, \; f \le 200 MHz$$
$$= 3.2 * [\log_{10}(11.75 * h_{re})]^2 - 4.97 dB, \; f \le 400 MHz$$

Equation 4-5

For a suburban and open areas the formula is modified as

Suburban:

$$L_{50} = L_{50}(urban) - 2 * [\log_{10}(f / 28)]^2 - 5.4 dB$$

Equation 4-6

Open areas:

$$L_{50} = L_{50}(urban) - 4.78 * (\log_{10} f)^2 - 18.33 * \log_{10} f - 40.98 dB$$

Equation 4-7

4.10.7 The POCAT Algorithm

The algorithm used by POCAT can be summarized as follows:

1. Determine the set $\boldsymbol{D_i}$ and the set of visible sectors $\boldsymbol{S_i}$ for each user.

2. Determine the noise associated with each base station η_k.
3. Determine the target carrier to interference ratio (CIR) γ. In POCAT, this value is the same for all users.
4. Calculate the path gain ***h*** between each user and each base station. The path gain is the same to all sectors in a cell, since they are all at the same distance from the user.
5. Calculate the transmitted power and sector assignment at time ***t*** for each user using equation (4-8) and (4-9).
6. Repeat Step 5 until the power vector converges to a solution

$$p_i(t+1) = \min \frac{\gamma_i * I_{ik}}{h_{ik}}, \text{ for all } k \in D_i \qquad \text{and } (s \in S_i) \in k$$

Equation 4-8

$$I_i^{(k,s)} = \sum_{j \neq i, s \in S_j} (h_{jk} * p_j(t)) + \eta_k \text{ , for all } j \in M$$

Equation 4-9

Equation (4-8) is basically the same as equation (4-1) and (4-2), except that equation (4-9) is used to calculate the total received power for sector ***s*** in cell ***k***, not including the power received from user ***i***. Equation (4-8) is calculated for all sectors that are visible for user ***i*** and that belongs to a cell ***k*** which belongs to the set $\boldsymbol{D_i}$.

POCAT can use a couple of different methods to select the set ***D***. The first one is to select the ***x*** closest base stations, where ***x*** could be any number between 2 and the total number of base stations in the network ***K***. The second method is to have a maximum distance between the user and a possible base station.

4.10.8 The Iteration process

The algorithm repeats Step 5 until the power vector converges to a solution. The question is when to stop this iteration process. In POCAT, the maximum number of iterations can be set as well as a maximum power difference to determine when to stop. The iteration process stops, when the total power difference between two iterations for all users is less than the parameter called “maximum power difference,” or if the maximum number of iterations is reached. Both of these values can be set before starting a simulation.

$$difference = \sum_{n=1}^{N} abs(p_n[i] - p_{n-1}[i-1])$$

Equation 4-10

In [39] tests have been made that show that 20 iterations is often as effective as 100 and that 10 is often sufficient. This is of course depending on the number of users and their locations.

4.10.9 Mobile Transmitters Maximum Output Power

In IS-95 standard [45], the mobile transmitters are classified into three different classes. The maximum output power for the mobile transmitter has to be within specified limits, depending on what group the mobile belongs to. See Table 6.

Table 6. Maximum Output Power

Mobile Station Class	Maximum Output Shall Exceed	Maximum Output Shall not Exceed
I	1 dBW (1.25 Watts)	8 dBW (6.3 Watts)
II	-3 dBW (0.5 Watts)	4 dBW (2.5 Watts)
III	-7 dBW (0.2 Watts)	0 dBW (1.0 Watts)

4.11 Implementation of POCAT

This section describes some implementation issues for the C version of POCAT and also the integration of POCAT with UTMOST written by Heidi McClure [46]. The first version of POCAT is implemented in C and is meant to be used as a tool that can calculate channel and power assignments on previously created traffic data; or as a tool that runs simultaneously with UTMOST and communicate via a file read and write interface.

4.11.1 How to use POCAT

POCAT is a command line based application with the following syntax:

pocat <YYMMDDHHMMSS> [/OPTIONS]

where YYMMDDHHMMSS is the Date_Time_Stamp for the simulation data created by the user traffic modeling tool. Before POCAT can be executed, the file globals.txt must be edited to make sure that the paths to the activity and topology files are correct. This file must be in the same directory as the POCAT executable. A more detailed description of POCAT and how to run it can be found in the POCAT man page in Section 10.

4.11.2 Files used by POCAT

POCAT reads the network information from two kinds of files created by UTMOST: one is a topology file and the other one is an activity file.

The topology file, called sector_data, consists of information about the set of sector antennae in the wireless network model. For each antenna, its location and its direction are listed.

The activity file (step-file) consists of a snapshot of the network. In each step-file, the user id, the location of the user, the speed of the user, the direction of the user, the users transmitted power, calling status (Calling, Roaming or Driving), assigned antenna, and visible sector antennae are listed. The activity files are named according to the

number of the step which was executed. That is, step one's results will be in a file called "1", step two's in "2."

The topology file and the activity file are both placed in a directory named according to the date and time (YYMMDDHHMMSS) of the UTMOST simulation run in which the files were created. See Figure 7.

Detailed description of each file generated or used by POCAT may be found in Section 0 and the documentation called "pocatfiles.txt" in the racewin directory.

4.11.3 Interaction with UTMOST

POCAT can communicate with UTMOST via file read/write. For each <#> step-file created, UTMOST also creates an empty <#>done file. By looking for the next <#>done created in the activity directory, POCAT knows when a new step-file is created and closed. After finding a new file, POCAT uses the data in this step file to generate a new power and channel assignment result file. The result file is named after the same number as the <#> step-file. POCAT stores this file plus an empty <#>done file in the chan_assign directory.

Example of interaction between UTMOST and POCAT:

UTMOST:

1. UTMOST creates the step-file “1”, after closing this file, it creates another file called “1done”.
2. UTMOST looks for a file called “1done” file in the chan_assign directory.
3. After finding the “1done” file, UTMOST reads the result file “1” and displays the assignments.

POCAT:

1. POCAT waits for the “1done” file to be created in the activity directory.
2. POCAT reads the data in “1” file and computes the new assignments.

POCAT saves the new assignments in file called “1” in the chan_assign directory and after closing this file, also creates a “1done” file in the same directory.

4.11.4 Calculating Visible sectors with sector overlap

To calculate the visible sectors for a user, POCAT uses the base station’s sector antenna direction and then figures out what direction this user is in relation to the base station. This method depends on the sector's direction and the angle from the base station to the users location point (x,y). A sector's direction attribute indicates the center of the 120 degree range of the sector antenna. This means the range visible by the sector is from (direction - (60 degrees + overlap)) to (direction +(60+overlap)). In POCAT, the three sector directions are 30°, 150° and 270°. Figure 19 shows the overlap for the sector with direction 270 degrees.

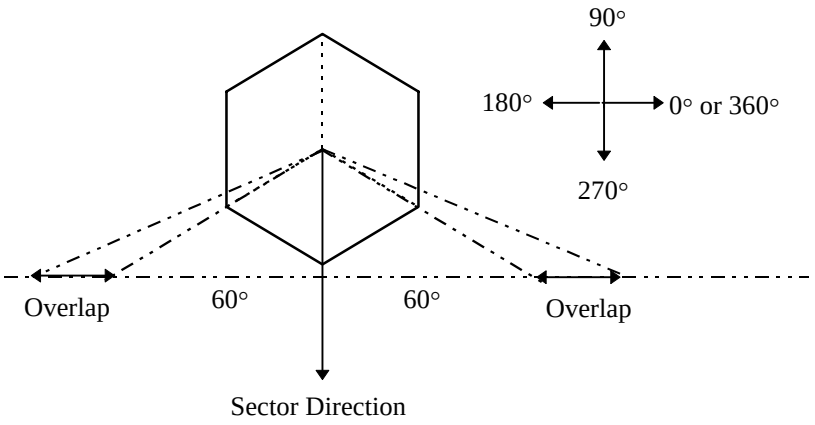


Figure 19. Sector Direction

The overlap angle is by default set to 15 %, i.e., 18°. By setting the overlap angle to 120°, the network will act like it is using omni-directional antennae. That is, all users will be seen by all sectors in the network.

4.12 Integrating POCAT with RACEWIN

The standalone C version of POCAT was ported to Java and was integrated with UTMOST. A couple of new features were also added to the simulator part of RACEWIN that were not implemented in UTMOST. These new features are explained here.

4.12.1 The Go Back feature

This makes possible to go back and forth through the simulated steps. The "Go" button in UTMOST has been divided into three buttons, one for creating a new step "→", one for going back to the previous step "←", and another one for recalculating the power and channel assignment for the current step "GO". The "GO" button is useful for comparing the results using different power control settings with the same network snapshot. Setting such as sector overlap, the number of channels and power constraints can be changed to find an optimal configuration for the set of users. Figure 20 shows the simulation controls available in RACEWIN.

Figure 20. Simulation Controls

Under the “GO” button is a text window for specifying the number of steps to be simulated at a time. The message under this text window displays number of steps simulated so far. It will also show which step that is displayed, when stepping back. For example, if 15 steps have been simulated and the “←“ is used to go back and display step 10, then the message text will be “step 10 of 15”.

4.12.2 The load simulation data feature

Data from a previous simulation can be loaded into RACEWIN for verification purpose and/or to create more simulation steps. Selecting **Open** from the **File** menu will bring up a dialog box for entering the date of the simulation data to be loaded.

The model settings for each simulation is saved in a file called model_data. This file is used when loading simulation data and contains information, such as the cell pattern, the number of user, and the number of steps created, and is stored in the same directory as the activity files.

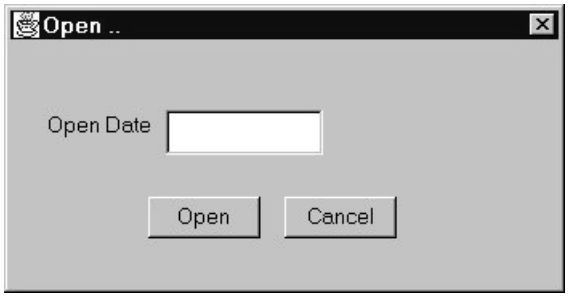


Figure 21. Open Dialog

4.12.3 Sector Overlapping

The sector overlap angle can be changed by using the sector properties in the Properties window.

4.12.4 Sector Information

Information about the sectors in the network can be displayed, such as the number of users assigned to the sector, the sectors total received power, and the total number of users dropped in the network. Dropped users are users that couldn’t be assigned to a base station due to the capacity limitation or power exceed limitations.

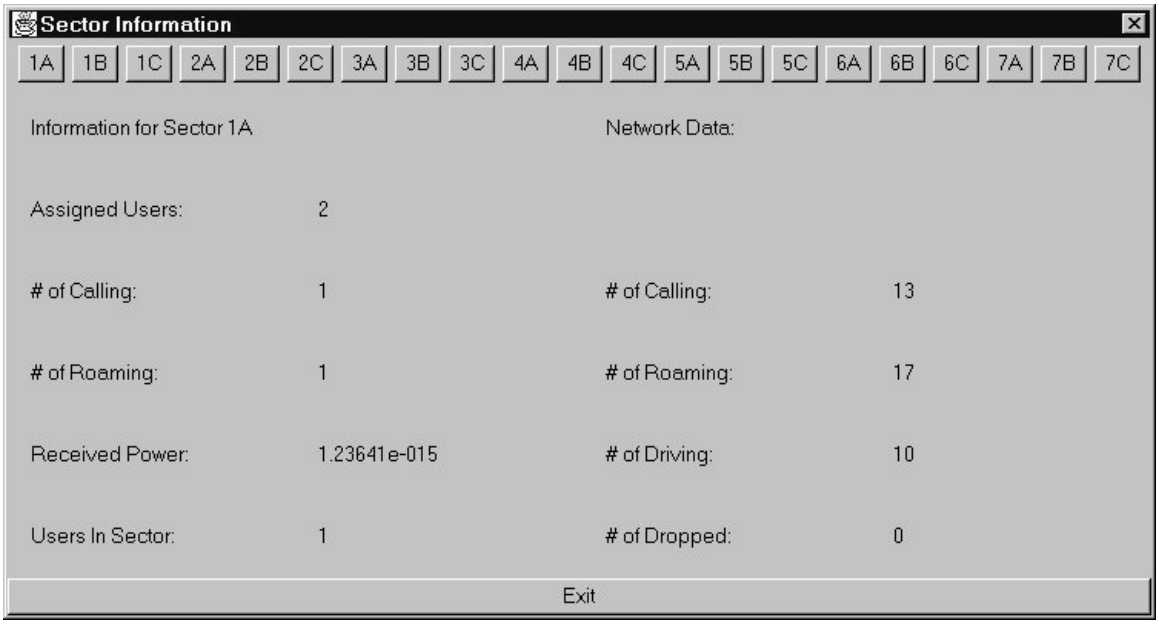


Figure 22: Sector Information

4.12.5 Background image

This feature is not yet fully implemented. The goal is to have the possibility to show a map such as GIS in the background and have the users follow streets instead of moving randomly. Right now a JPEG or GIF file can be loaded as background and the cell pattern can be moved around on top of the background.

4.12.6 Classes added

RACEWIN is built upon the classes created for UTMOST. A couple of classes were added to handle the features mentioned above. The classes added are: ImagePanel, LoadDialog, POCAT, POCATSettings, PowCtrlPanel, and SectorInfo

4.13 Porting POCAT from C to Java Experience

The similarities between C and Java made porting POCAT easier. One thing to be concerned about Java is the execution speed. The calculation during a simulation takes a lot of computation and it would be interesting to compare with a C++ version. Java is a really nice language to use and easy to learn. Microsoft's Visual J++ is used as a development tool. This tool was really helpful for debugging and the online help was nice. Problems occurred when the byte code generated by J++ was run by JDK:s Virtual Machine. To solve this problem, the source code needs to be recompiled using JDK:s compiler.

4.14 Execution Speed

The following test shows the execution time for RACEWIN to simulate one step with the number of users varies from 25 to 500. All users were placed randomly and the power control setting were set so that no users were dropped. The test was done on a

Pentium 166MHz machine with 64Mb memory running Windows NT 4.0 Workstation and JDK version 1.1.1.

As the number of users increases to over 200, the waiting time starts to get over 20 seconds and may be too long for practical use. The execution time includes the time it takes to calculate new user positions, to generate the boundary crossing information, and to calculate the power assignments.

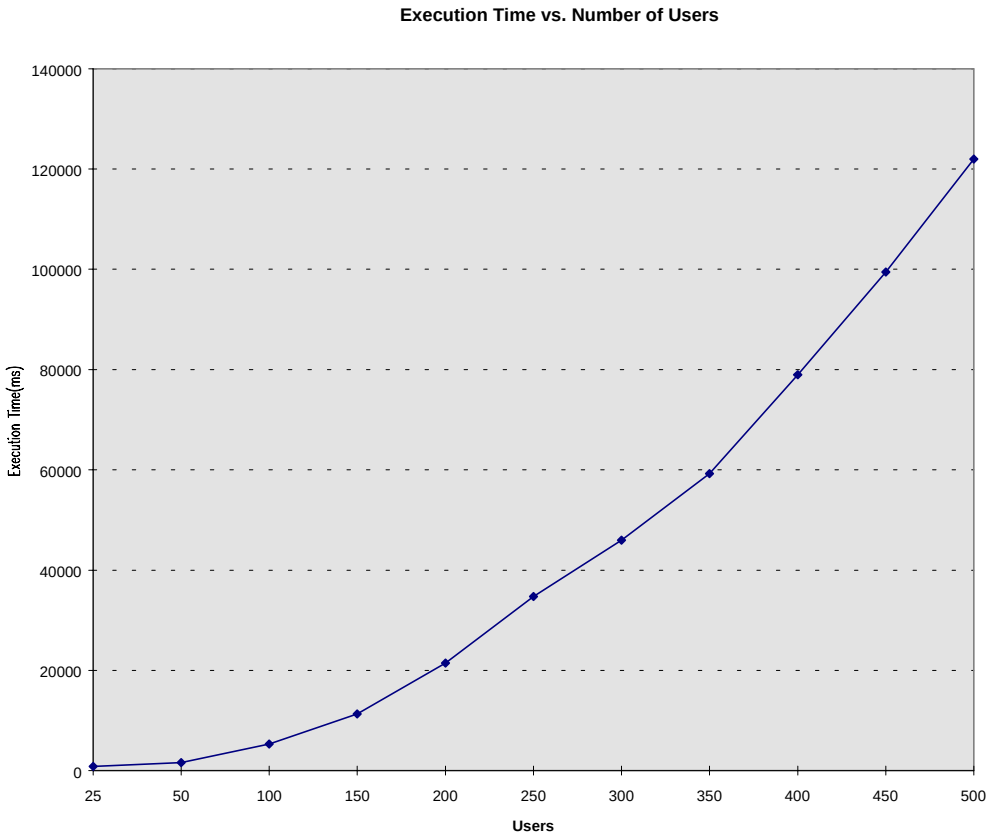


Figure 23. Execution Time

4.15 Verifying POCAT results

Here are a couple of different user snapshots to verify the power and channel assignment results generated by POCAT.

4.15.1 Test 1

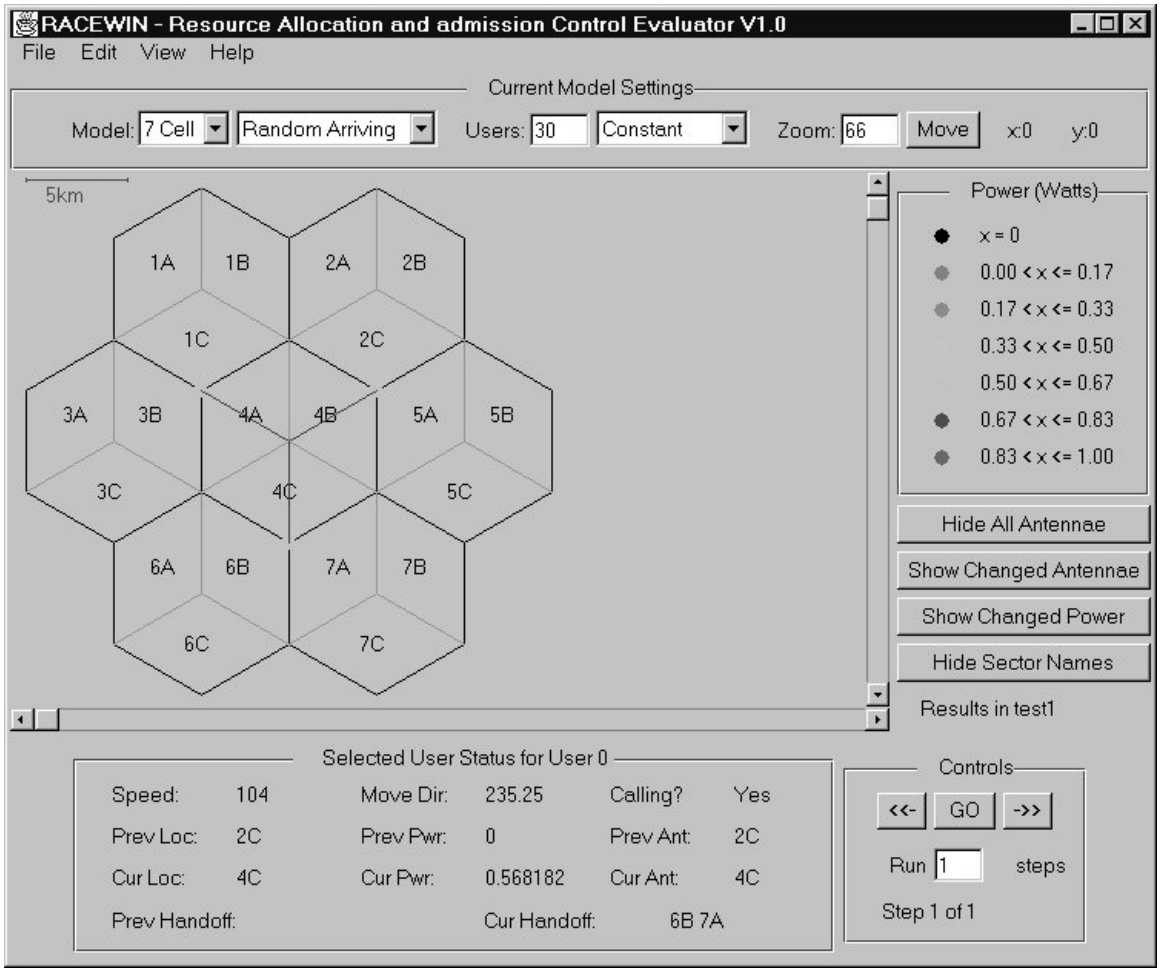


Figure 24. Test1

Test 1 uses the following data: 7-cell network with a cell radius of 5 km. 30 user were separated into three groups with 10 users in each of the corners, where sectors (1C, 3B ,4C), (2C, 4B, 5A) and (4C, 6B, 7A) intersect.

The users are all at the same distance, 5 km, from the three closest base stations, and since the external noise power is the same at all base-stations, in this case 1e-14 W, the assigned power should be the same for all of the users.

Of the three closest base stations at each user location, sectors 4A, 4B and 4C are the ones that receives the lowest interference power. Each of these sectors can “see” ten users. Therefore all users are assigned to the middle sectors.

This test indicates that the algorithm does not take into consideration of the traffic in the base station, it only addresses the interference power. All of the thirty users were assigned to the middle cell which means that all network traffic is going through that base station and the other six base stations have no traffic at all. This load balancing problem is a good example of the kind of problems RACEWIN can be used to identify.

Output from POCAT:

Users 0-9 are located at position 1, 10-19 at position 2, 20-29 at position 3. The number “3” in the new antenna column shows the number of antennae that the user is assigned to. In this example, user 0 is assigned to antenna 4C, but is also in handoff with 6B and 7A. The “0” in the old antenna column shows that the user was not assigned to any antenna in the previous step. This is because Step 1 is the first step simulated.

step	user i	old power	new power	old antennas	new antennae
1	0	0.000000	0.568182	0 1A	3 4C 6B 7A
1	1	0.000000	0.568182	0 1A	3 4C 6B 7A
1	2	0.000000	0.568182	0 1A	3 4C 6B 7A
1	3	0.000000	0.568182	0 1A	3 4C 6B 7A
1	4	0.000000	0.568182	0 1A	3 4C 6B 7A
1	5	0.000000	0.568182	0 1A	3 4C 6B 7A
1	6	0.000000	0.568182	0 1A	3 4C 6B 7A
1	7	0.000000	0.568182	0 1A	3 4C 6B 7A
1	8	0.000000	0.568182	0 1A	3 4C 6B 7A
1	9	0.000000	0.568182	0 1A	3 4C 6B 7A
1	10	0.000000	0.568132	0 1A	3 4A 1C 3B
1	11	0.000000	0.568132	0 1A	3 4A 1C 3B
1	12	0.000000	0.568132	0 1A	3 4A 1C 3B
1	13	0.000000	0.568132	0 1A	3 4A 1C 3B
1	14	0.000000	0.568132	0 1A	3 4A 1C 3B
1	15	0.000000	0.568132	0 1A	3 4A 1C 3B
1	16	0.000000	0.568132	0 1A	3 4A 1C 3B
1	17	0.000000	0.568132	0 1A	3 4A 1C 3B
1	18	0.000000	0.568132	0 1A	3 4A 1C 3B
1	19	0.000000	0.568132	0 1A	3 4A 1C 3B
1	20	0.000000	0.568132	0 1A	3 4B 2C 5A
1	21	0.000000	0.568132	0 1A	3 4B 2C 5A
1	22	0.000000	0.568132	0 1A	3 4B 2C 5A
1	23	0.000000	0.568132	0 1A	3 4B 2C 5A
1	24	0.000000	0.568132	0 1A	3 4B 2C 5A
1	25	0.000000	0.568132	0 1A	3 4B 2C 5A
1	26	0.000000	0.568132	0 1A	3 4B 2C 5A
1	27	0.000000	0.568132	0 1A	3 4B 2C 5A
1	28	0.000000	0.568132	0 1A	3 4B 2C 5A
1	29	0.000000	0.568132	0 1A	3 4B 2C 5A

4.15.2 Test2

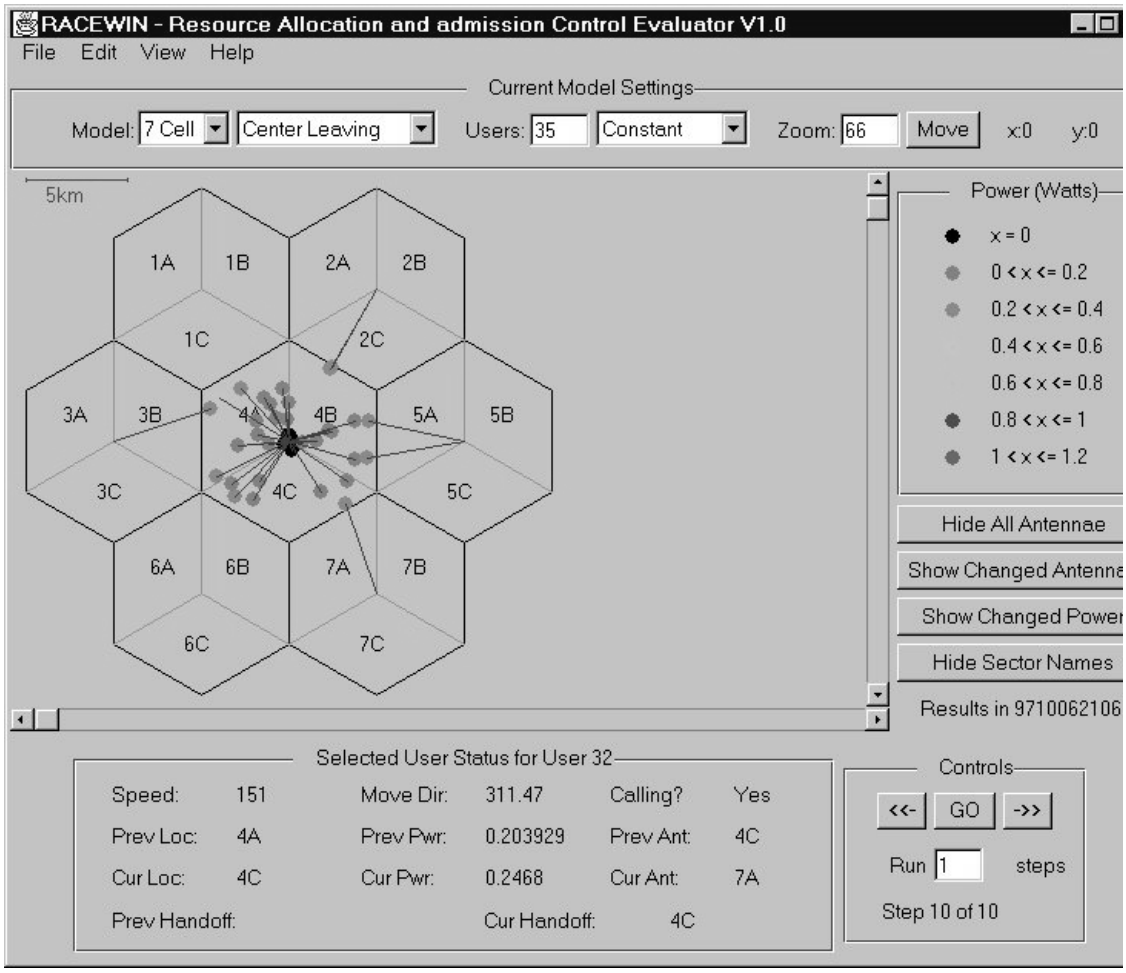


Figure 25. Test 2

7-cell network with a cell radius of 5 km. 35 users were placed at the center of cell 4 using a "center leaving" pattern. After 10 steps, some of the user at the edge of cell 4 are assigned to nearby sectors outside of cell 4.

step	user i	old power	new power	old antennas	new antennae
10	0	0.000005	0.000008	2 4C 4A	2 4C 4A
10	1	0.314082	0.224250	2 2C 4B	2 2C 4B
10	2	0.017795	0.023569	1 4B	1 4B
10	3	0.012660	0.016755	2 4B 4A	2 4B 4A
10	4	0.006964	0.009878	1 4A	1 4A
10	5	0.000001	0.000002	1 4B	1 4B
10	6	0.000300	0.000398	1 4B	1 4B
10	7	0.042107	0.059780	1 4A	1 4A
10	8	0.000007	0.000009	2 4B 4A	2 4B 4A
10	9	0.292779	0.258231	1 4B	2 2C 4B
10	10	0.168879	0.252446	2 4C 4A	2 4C 4A
10	11	0.089490	0.133833	2 4C 4B	2 4C 4B
10	12	0.145607	0.217809	1 4C	1 4C
10	13	0.300436	0.303833	1 4B	2 5A 4B

```
10   14 0.346994 0.371477  1 4A      2 3B 4A
10   15 0.023300 0.034868  1 4C      1 4C
10   16 0.263144 0.331723  1 4B      2 5A 4B
10   17 0.049241 0.065258  2 4B 4A   2 4B 4A
10   18 0.052640 0.078806  1 4C      1 4C
10   19 0.302848 0.429828  1 4A      1 4A
10   20 0.142291 0.188416  1 4B      1 4B
10   21 0.003552 0.004700  1 4B      1 4B
10   22 0.003144 0.004464  1 4A      1 4A
10   23 0.018605 0.026451  1 4A      1 4A
10   24 0.000039 0.000059  1 4C      1 4C
10   25 0.046281 0.065676  1 4A      1 4A
10   26 0.013340 0.018937  1 4A      1 4A
10   27 0.001053 0.001494  1 4A      1 4A
10   28 0.092000 0.137652  1 4C      1 4C
10   29 0.101482 0.151802  2 4C 4B   2 4C 4B
10   30 0.171257 0.243161  1 4A      1 4A
10   31 0.023351 0.030918  1 4B      1 4B
10   32 0.203929 0.246800  1 4C      2 7A 4C
10   33 0.104186 0.155926  2 4C 4A   2 4C 4A
10   34 0.001117 0.001478  2 4B 4A   2 4B 4A
```

4.15.3 Test 3

This test shows the result of having more users inside a sector than the number of available channels in the sector. The number of channels available in a sector is the maximum number of users that can be assigned to the sector. This number can be changed in the sector properties.

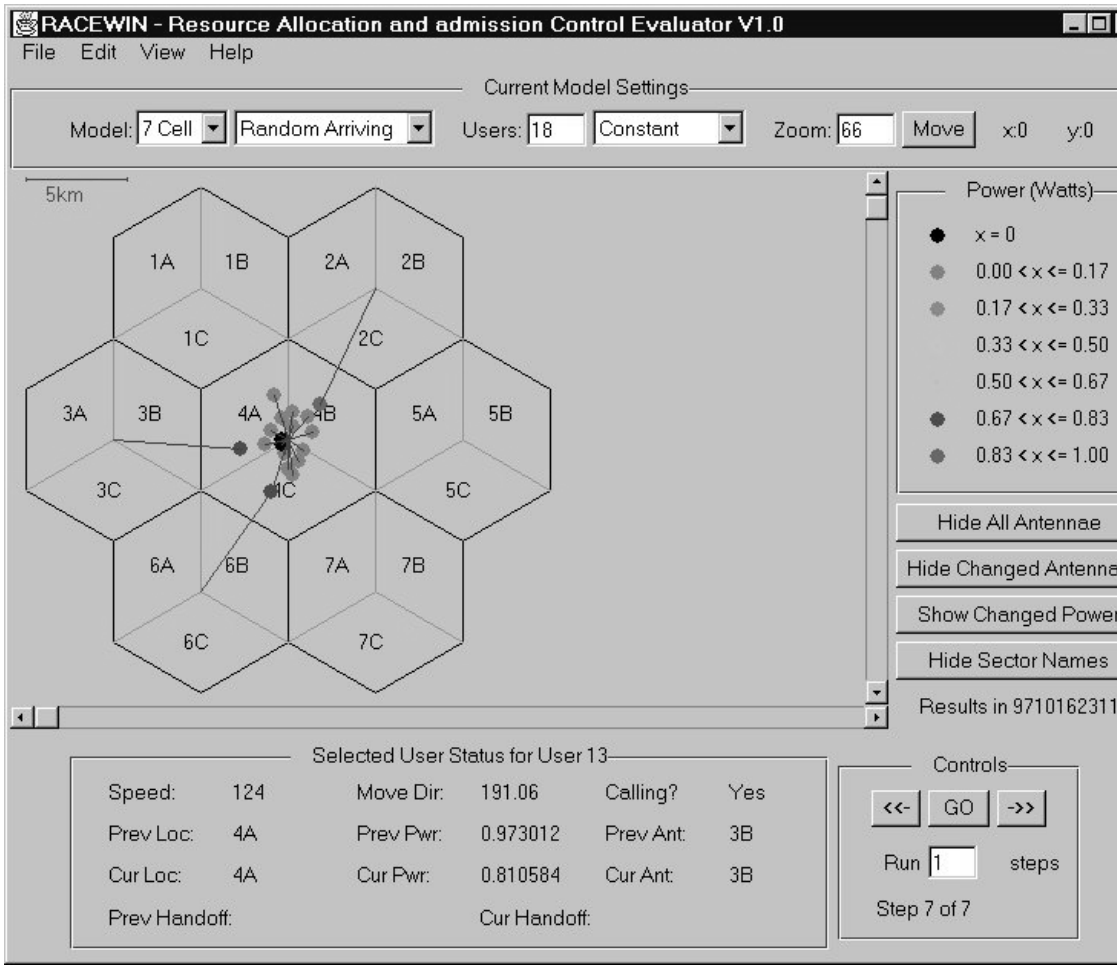


Figure 26. Test 3

In this example, 18 users are placed in the center of cell 4, as in test 2. Here the number of available channels for each sector is set to 5, instead of the default value of 40.

Since all users are in the center of cell 4, only 15 of them can be assigned, but as the users leave the center of the network, the three users, that were dropped in the fist couple of steps, are assigned to nearby cells. See Figure 26.

4.16 Future improvements of POCAT and RACEWIN v1.0

Here we discuss where POCAT and RACEWIN v1.0 may be extended, improved or enhanced.

4.16.1 Modify POCAT to consider Network Traffic

Make the algorithm consider the network traffic. In "Test 1" all the traffic goes through the center cell and none through the nearby cells. A decision needs to be made on what is more important; low interference or load balancing among base stations in the network.

4.16.2 Overlay on GIS Map

Extend the feature to load a picture (map) in the background with the possibility to read specific road and geological features from a GIS database. This requires the user traffic model to be extended and allows the road and traffic situation to read in from the GIS database. To make the cell placement more realistic, each cell needs to be placed separately and it is not necessary to follow the rigid cell pattern.

Integrating with the 3D geology information will allow us to study the coverage of the set of cell antennae in a geographical area by including models of multipath signal computation. By including the population/user information in the model, it will allow more realistic simulation and the simulation study for future network planning.

4.16.3 Integrate with Wireless Network Discrete Event Simulator for QoS studies

A preliminary discrete event simulator, called RACESIM, for wireless network was designed and implemented by James Haring and Edward Chow. It tried to model the processing power of the base stations and mobile switching office, the bandwidth of connections among them, the propagation delay, and the message exchange sequence. It takes the input generated by the UTMOST as events, simulates the message processing in the wireless network, and computes the Quality of Service parameters, such as the number of users served in the area, the number of connections lost due to the long handoff, and the number of users not admitted. POCAT can be integrated into the RACESIM for power and cell site assignments. The RACESIM simulation results can be read back by the RACEWIN GUI to facilitate the wireless network design.

4.16.4 Batch Mode Operation

To speedup and simplify the process of creating simulation data, a command line interface can be implemented that allows us to run simulation without displaying the RACEWIN graphical user interface. Properties such as the overlap angle, the number of users and the number of steps could be entered from the command line. The simulation data would be created as before, but without displaying each step graphically. With this batch mode, scripts can be written to facilitate the study of the impact of certain network parameters.

5. Evaluation

The success of the proposed project can be assessed by the usefulness and performance of software modules provided by the RACEWIN system. Based on the results presented in Section 4, the project is quite successful, since we now have a graphical user interface for visualizing and analyzing the wireless user traffic patterns, and power and cell site assignment results. We have a user traffic modeling and simulation tool, called UTMOST, that can generate traffic events and data for analyzing the optimization packages for power and cell site assignments. The data generated can also be used for driving the discrete event simulators that model the wireless network resources, admission control and handoff procedures, and collect statistics on the QoS for cell size determination and network planning or improvement We also implemented an efficient power and cell site assignment Tool, called POCAT, that reduces the interference by minimizing the total user transmission power. These tools can assist network planners or managers to plan or evolve wireless networks. They augment the library of tools that can foster the research, education, and development efforts in the area of wireless network planning and traffic analysis.

US West will provide feedback on the RACEWIN usage in their research and network management organizations and the usefulness of the report. Those feedback will further indicate the degree of success of the project.

6. Intellectual property developed under sponsorship of this grant.

We have designed and implemented RACEWIN software system with modules that can be used to generate and graphically display the wireless network traffic data for a variety of network models and traversing patterns, and to verify power assignment results. They can be licensed to companies in telecommunications industry that operate or plan wireless networks. To obtain the Java byte code or source code of the RACEWIN system, send email to chow@cs.uccs.edu.

7. Technology Transfer

We have given presentations of our research results to researchers at US West Advanced Technologies twice and got valuable feedback from them. Based on that, we have emphasized on the design of user traffic modeling and simulation tool for visualizing and verifying user traversing patterns and power assignment results. They helped us identify the network models and traversing patterns. They also brought to our attention the need for modeling the soft handoff and the sector overlap. We have delivered the RACEWIN software to US West Advanced Technologies.

Acknowledgment

We would like to thank Dr. Steve Chiu and Dr. Jenny Sanchez for initiating and guiding the project in their initial phase, Dr. Subramanian Vasudevan and Prof. Roger Ziemer for providing information about wireless information network. We also would like to thank Dr. Sherwin Wang and Justin Chuang for answering tedious questions on wireless network equipment and signal fading models. Mark Petzod provided a very useful report on signal fading models and existing dynamic channel assignment work.

8. References

1. FCC, “Amendment of the Commission’s Rules to Establish New Personal Communications Services,” GEN Docket No. 90-314, Oct. 22, 1993.
2. FCC, “Amendment of the Commission’s Rules to Establish New Personal Communications Services,” GEN Docket No. 90-314, June 13, 1994.
3. “1994 PCS Market Demand Forecast,” Personal Communications Industry Association, Washington, DC, Jan. 1994.
4. Lon-Rong Hu and S. S. Rappaport, “Personal Communications Systems Using Multiple Hierarchical Cellular Overlays,” IEEE JSAC, Vol. 13, No. 2, Feb. 1995, pp. 406-415.
5. Chih-Lin I, L. J. Greenstein, and R. D. Gitlin, “A Microcell/Macrocell Cellular Architecture for Low- and High-Mobility Wireless Users,” IEEE JSAC, Vol. 11, No. 6, Aug. 1993, pp. 885-891.
6. Mischa Schwartz, “Network Management and Control Issues in Multimedia Wireless Networks,” IEEE Personal Commun., Vol. 2, No. 3, June 1995, pp. 8-16.
7. Vinko Erceg and J. F. Whitehead, “Microcell Size and Architecture Analysis From the Propagation and Capacity Point of View,” Proceedings of IEEE 44th Veh. Tech. Conf., VTC 94, pp. 215-218.
8. Miltiades E. Anagnostou, G. C. Manos, “Handover Related Performance of Mobile Communication Networks,” Proceedings of IEEE 44th Veh. Tech. Conf., VTC 94, pp. 111-114.
9. Rajiv Vijayan and J. M. Holtzman, “A Model for Analyzing Handoff Algorithms,” IEEE Trans. on Vehicular Technology, August, 1993.
10. Djamal Zeghlache, “Aggressive Handover Algorithm for Mobile Networks,” Proceedings of IEEE 44th Veh. Tech. conf., VTC 94, pp. 87-90.
11. C.-H. Chow, “On Multicast Path Finding Algorithms,” Proceedings of INFOCOM’91, pp. 1274-1283, Miami, Florida, April 7-11, 1991.
12. C.-H. E. Chow, J. Bicknell, S. McCaughey, and S. Syed, “A Fast Distributed Network Restoration Algorithm,” Proceedings of 12th International Phoenix Conference on Computers and Communications, March 24-26, 1993, Scottsdale, Arizona.
13. C.-H. E. Chow, J. Bicknell, and S. Syed, “Performance Analysis of Fast Link Restoration Algorithms,” Journal of Digital and Analog Communication Systems, 1995. Part of the research results published in Proceedings of Globecom 93.
14. C.-H. E. Chow, J. Bicknell, S. McCaughey, and V. Narasimhan, “NETRESTORE: A network simulation system,” to be published in 1996 on Journal of Computer and Software Engineering.
15. C.-H. E. Chow, “NETSIM: A network simulation system for the design and analysis of network survival techniques,” CASI proposal, November, 1993.
16. C.-H. E. Chow, “Resource Allocation for Multiparty Connections,” IFIP 3rd International Workshop on Protocols for High-Speed Networks, May 1992, pp. 121-136.
17. C.-H. E. Chow, “ATMROS: A Network Design System for ATM Network Optimization and Traffic Management,” CASI proposal, November, 1994.

18. FCC, “Amendment of the Commission’s Rules to Establish New Personal Communications Services,” GEN Docket No. 90-314, Oct. 22, 1993.
19. Mischa Schwartz, “Network Management and Control Issues in Multimedia Wireless Networks,” IEEE Personal Communications, Vol. 2, No. 3, June 1995, pp. 8-16.
20. Leung, Kin K., “Traffic Models for Wireless Communications Networks,” IEEE JSAC, Vol. 12, No. 8, Oct. 1994.
21. Miltiades E. Anagnostou, G. C. Manos, “Handover Related Performance of Mobile Communication Networks,” Proceedings of IEEE 44th Veh. Tech. Conf., VTC 94, pp. 111-114.
22. Djamal Zeghlache, “Aggressive Handover Algorithm for Mobile Networks,” Proceedings of IEEE 44th Veh. Tech. Conf., VTC 94, pp. 87-90.
23. Rajiv Vijayan and J. M. Holtzman, “A Model for Analyzing Handoff Algorithms,” IEEE Trans. On Vehicular Technology, August, 1993.
24. Lon-Rong Hu and S. S. Rapport, “Personal Communications Systems Using Multiple Hierarchical Cellular Overlays,” IEEE JSAC, Vol. 13, No. 2, Feb. 1995, pp. 406-415.
25. Chih-Lin I, L. J. Greenstein, and R. D. Gitlin, “A Microcell/Macrocell Cellular Architecture for Low- and High-Mobility Wireless Users,” IEEE JSAC, Vol. 11, No. 6, Aug. 1993, pp. 885-891.
26. Katzela, I. and M. Naghshineh, *Channel Assignment Schemes for Cellular Mobile Telecommunication Systems: A Comprehensive Survey*, IEEE Personal Communications, June 1996, pp. 10-31.
27. Lemay, Laura and Charles L. Perkins, *Teach Yourself Java in 21 Days*, Sams.net Publishing, 1996.
28. Geary, David M. and Alan L. McClellan, *Graphic Java - Mastering the AWT*, The Sunsoft Press, 1996.
29. Jackson, Jerry R. and Alan L. McClellan, Java By Example, The Sunsoft Press, 1996.
30. Cornell, Gary and Cay S. Horstmann, *Core Java*, 2nd Edition The Sunsoft Press, 1997.
31. van der Linden, Peter, *Just Java*, 2nd Edition, The Sunsoft Press, 1997.
32. Foley, James D., et. al. *Computer Graphics, Principles and Practice*, 2nd edition, 1990, Addison-Wesley Publishing Company, Inc.
33. Hodgman, Charles D., *Mathematical Tables from Handbook of Chemistry and Physics*, 7th edition, 1941, Chemical Rubber Publishing Co. Cleveland, Ohio.
34. Heller, Paul, Instructor for an advanced Java programming class offered by Sun Microsystems in Denver, March 3 - 8, 1997.
35. Web Page: National Geologic Map Database, USGS, National Cooperative Geologic Mapping Program, National Geologic Mapping Act of 1992.
36. Brown, Ronald H, et. al. TIGER/Line (TM) Files, 1992, Technical Documentation, Washington, D. C. 1993, U. S. Department of Commerce.
37. Lee, William C. Y., *Mobile Cellular Telecommunications*, 2nd Edition, McGraw-Hill, Inc., 1995
38. Stephen V. Hanly, “An Algorithm for Combined Cell-Site Selection and Power Control to Maximize Cellular Spread Spectrum Capacity”, IEEE Journal on Selected Areas in Communication, VOL 13. NO 7. September 1995

39. Roy D. Yates and Ching-Yao Huang, “Integrated Power Control and Base Station Assignment”, IEEE Transactions on vehicular Technology, VOL. 44, NO. 3, August 1995.
40. I. Katzela and M. Naghshineh, “Channel Assignment for Cellular Mobile Telecommunication Systems: A Comprehensive Survey”, IEEE Personal Communications, June 1996.
41. Rappaport, T. S., “Wireless Communications: Principles and Practice” Upper Saddle River, New Jersey: Prentice Hall, 1996
42. K. Feher, “Wireless Digital Communications”, Upper Saddle River, New Jersey: Prentice Hall, 1995.
43. K. Pahlavan and A. H. Levesque, “Wireless Information Networks”, Wiley-Interscience, John Wiley & Sons, 1995.
44. Jerry D. Gibson, “The Mobile Communications Handbook”, CRC Press (Published in Cooperation with IEEE Press), 1996.
45. TIA/EIA Interim Standard, “Mobile Station-Base Station Compatibility Standard for Dual-Mode wideband Spread Spectrum Cellular System”, Telecommunications Industry Association, July 1993.
46. M. Heidi McClure, “UTMOST: A Traffic Modeling Tool in Java”, Master Thesis from University of Colorado at Colorado Springs, May, 1997.
47. V. K. Garg and J. E. Wilkes, “Wireless and Personal Communications Systems” Prentice Hall, 1996.

9. Appendix 1: Man Page for UTMOST

NAME

utmost - User Traffic MOdeling and Simulation Tool.

SYNOPSIS

java [-D <DEBUG_LEVEL>] utmost

DESCRIPTION

The utmost command starts the UTMOST tool. The tool may be started using one or more debug directive which will print various debug and diagnostic information to the console.

UTMOST is a user traffic modeling and simulation tool intended for PCS Wireless Information Network simulations. The tool generates data for various traffic patterns. From one (1) to five thousand (5000) mobile users may be modeled.

A seven cell circular pattern and a four cell linear pattern are provided. The seven cell circular model simulates the events where mobile users gather at the central cell (e.g. a football stadium). The four cell linear model simulates highway traffic.

Arriving, leaving and random patterns may be selected. The users may start randomly or may start from the perimeter or center of the model. Users may be constant in the model or they may regenerate after reaching their destination.

Constant user behavior means that once a user is initialized, that user never changes its identity. That is, if the user travels outside the boundaries of the model, the user just continues on its path, or if the user reaches its destination in the center of the model, the user just stays at the center.

Regenerating user behavior means that once a user reaches its destination this user is re-initialized. That is, the user is given totally new coordinates, power levels, and direction of travel.

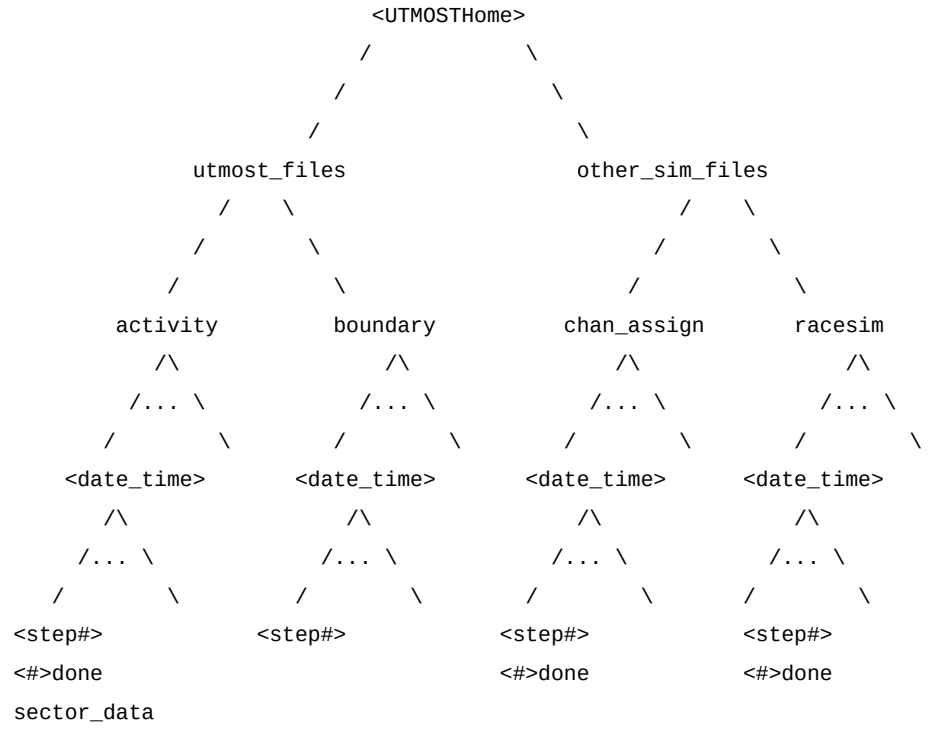
Once settings for the model are selected, the "Go" button is pressed to initiate 1 or more steps in the model. Two directories are created for each simulation run. An "activity/<date_time_stamp>" directory contains information for each user at each step of the simulation. This directory also

contains the sector_data used for this particular simulation run.

Another directory called "boundary/<data_time_stamp>" contains information about any user who has crossed a sector boundary at each step of the simulation. Results from each simulation step are stored in separate files with applicable users reported in that file.

The files in both the "activity/<data_time_stamp>" and the "boundary/<data_time_stamp>" directories are named the number of the step which was executed. That is, step one's results will be in a file called "1", step two's in "2", etc.

Below is a visual description of the file hierarchy:



Where,

utmost_files, other_sim_files, activity, boundary, chan_assign, and racesim are hard coded pathnames. other_sim_files, chan_assign, and racesim are not part of UTMOST at this time. They are results from other discrete event simulators.

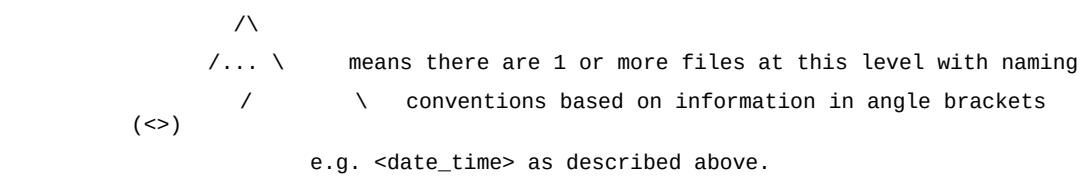
<UTMOSTHome> is the path to the UTMOST class files e.g. ~racewin/src.

<date_time> is a YYMMDDHHMMSS date time stamp, e.g. 970222160528.

<step#> is the current step in the simulation.

<#>done is an empty file created after the <step#> file has been closed.

sector_data is a file created with this run's sector data information.



means there are 1 or more files at this level with naming conventions based on information in angle brackets (<>)

e.g. <date_time> as described above.

Detailed description of each file generated or used by UTMOST may be found in the file with the documentation called "datafiles" or in the user's guide.

OPTIONS

The following options to the "java utmost" command are supported:

-D <DEBUG_LEVEL>

DEBUG_LEVEL specifies which debug statements are printed to the console. The following DEBUG_LEVEL's are supported:

DEBUG	shows all debug comments
DEBUG_ALL	shows all debug comments
DEBUG_SECTOR	shows the sector initialize comments
DEBUG_FLOW	shows some general program flow comments
DEBUG_MOVE	shows some information on calculating initial 7 cell arrive directions and general next position calculations
DEBUG_SEGMENT	shows segment setup comments
DEBUG_INTERSECT	shows intersection flow comments
DEBUG_USER	shows user update comments
DEBUG_VISIBLE	shows info about calculating the visible sector antennae to a user
DEBUG_INSIDE	shows info about the polygon inside function.

SEE ALSO

The UTMOST User's Guide; the "datafiles", 4_sector_data, and 7_sector_data files in the UTMOST Documentation directory. (4_sector_data and 7_sector_data should also be in the <UTMOST_HOME> directory)

AUTHOR

M. Heidi McClure, University of Colorado, Colorado Springs,
who may be reached at mhmcclur@elbert.uccs.edu or at
heidi.mcclure@sun.com.

HISTORY

Initial release: 7 Apr 1997
Latest revision: 24 Apr 1997

10. Appendix 2: Man page for POCAT

NAME

pocat - Power control and Channel Assignment Tool.

SYNTAX

pocat <YYMMDDHHMMSS> [/OPTIONS]

<YYMMDDHHMMSS> is the date_time_stamp directory where the activity data can be found
Example: pocat 970311114752

The Options Section (below) explains available options.

DESCRIPTION

POCAT is a power control and channel assignment tool for PCS Wireless Information Network simulations and is intended to use with UTMOST, which is a user traffic modeling and simulation tool.

POCAT reads traffic pattern data generated by UTMOST and implements two different algorithms for assigning the users with an optimal cell-site and up-link power. The algorithms are one by Hanly and one by Yates and Huang (MPA). See POCAT documentation for more information about these different algorithms.

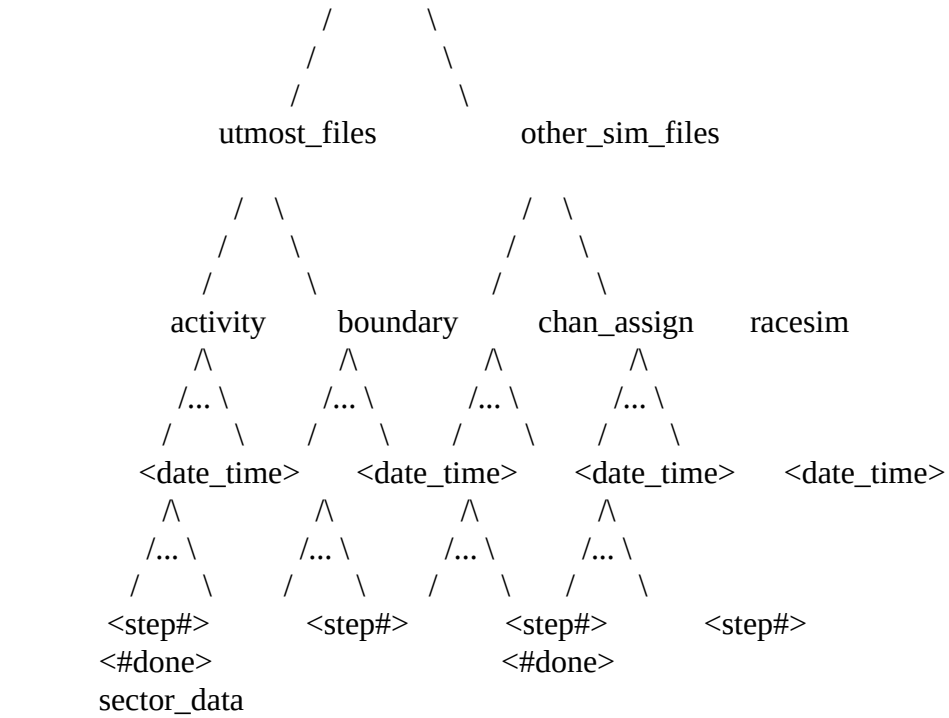
The “activity/<date_time_stamp>” directory is the directory containing the information for each user at each step of the simulation. This directory should also contain the sector_data file used for this particular simulation run. From one (1) to five thousand (5000) mobile users may be modeled in a network. The network can consist of a maximum of seven (7) cells, where each cell contains three sectors.

For each run of POCAT, a directory called “chan_assign/ <date_time_stamp>” is created, which contains the power and channel assignment for each step.

The files in both the "activity/<data_time_stamp>" and the “chan_assign/<date_time_stamp>” directories are named after the number of the step which was executed. That is, step one's results will be in a file called "1", step two's in "2", etc. For each file, a #done file is created when the file is “done” and closed, e.g., for the file called “1,” a “1done” file is created.

Below is a visual description of the file hierarchy:

<UTMOSTHome>



UTMOSTHome is the directory in which the UTMOST program is found. This can be changed as described in the Options section.

The default values of some of the global variables used by POCAT can be changed by editing the file “global.txt”. This file should be located in the same directory as “pocat.exe”.

Detailed description of each file generated or used by POCAT may be found in the file with the documentation called "pocatfiles.txt" or in the user's guide.

OPTIONS

Options and parameters supported by POCAT are listed below.

ALGORITHM- Selects which algorithm to use. The default algorithms is Hanly’s (0). It can be changed to the MPA by using: /a=1

ALLOWED BASE STATIONS- This option is used to change the method use to select the base stations that a user can be assigned to (Hanly’s algorithm). To select the three(3) closest (2-4 is allowed), use /b=3. To select the base stations that are within a distance of 7500 meters, use /bm=7500.

CIR- The minimum Carrier to Interference ratio for the mobile user is used in MPA. To set CIR to 0.05, use /c=0.05.

DEBUG- This option is used to select debug mode. /d=1 will make POCAT print more debug information.

FADING- This option specifies what type of propagation path loss formula to use. Default is a single step fading law with fading factor=4.

To select
Single step with factor 3: /fs=3
Multiple step: /fm
To set: short distance to 1500 meters (default =): /fms=1500
medium distance to 4000 meters (default =): /fmm=4000
friis equation: /ff
To set: Transmitter gain: /fft=2
Receiver gain: /ffr=4

Hata model: /fh (default = small to medium city)
To set: Large city /fhl
Suburban /fhs
Open areas /fho

NOISE- To set the Noise power at the base stations to 1e-10 use /n=1e-10.

REPEATS- Maximum number of iterations for the selected algorithm is set to 20 (default=15) by using /r=20. The number of iterations is also limited by the difference in total transmitted power between two iterations. To set this value use e.g. /rd=1e-10 (default = 1e-8).

STEP- This option can be used to start at a different step number then one (1). To start at step 60 use /s=60.

TRANSMITTER CLASS- The transmitter class is used to define the mobile transmitter's maximum and minimum power levels according to IS-95. There is three (3) different classes of transmitters to choose from. To select class two (2) use /t=2 (default =1). For more information about the different classes see POCAT documentation

SEE ALSO

The POCAT User's Guide and the "pocatfiles.txt" and the file "global.txt".

AUTHOR

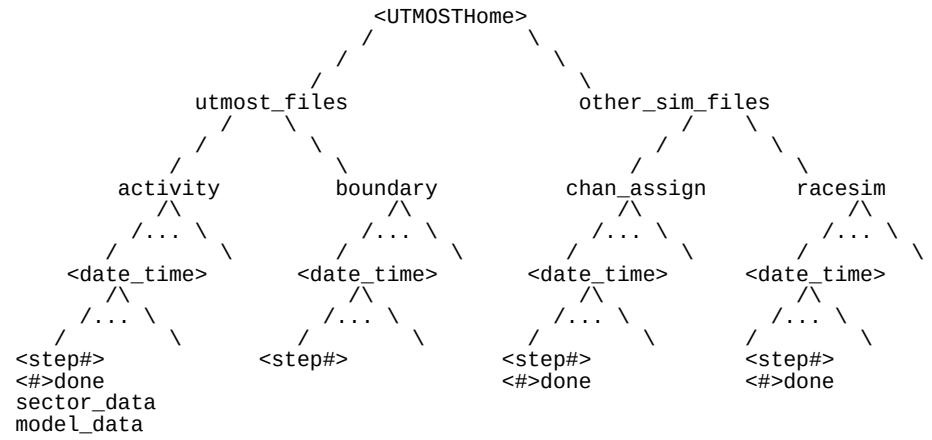
Jonas Hedlind, University of Colorado, Colorado Springs,
who may be reached at jehedlin@mail.uccs.edu.

HISTORY

Initial release: 25 April 1997
Latest release: 12 May 1997

11. Appendix 3: Data Files used by POCAT and RACEWIN

```
================================================================================
Datafiles created by or used by POCAT and RACEWIN.
This file is a modified copy of the datafiles file for UTMOST.
================================================================================
NOTE that this file is best viewed with a fixed font like courier.
================================================================================
Directory trees
```




```
Where,
    utmost_files, activity and chan_assign, can be changed by editing the
   globals.txt file.
    <UTMOSTHome> is the path to the UTMOST class files e.g. ~racewin/src.
    <date_time> is a YYMMDDHHMMSS date time stamp, e.g. 970222160528.
    <step#> is the current step in the simulation.
    <#>done is an empty file created after the <step#> file has been closed.
    sector_data is a file created with this run's sector data information.

        /\
      /... \     means there are 1 or more files at this level with naming
     /      \    conventions based on information in angle brackets (<>) e.g.
                 <date_time> as described above.


Each <step#> file contains information about all users at that current step.
In some cases, like in boundary data, there may be no information.  The file
should still be created, but will have nothing in it.

<#>done files are only created in the activity directories.

sector_data files are only created in the activity directories.  These files
are currently just a copy of the 7_cell or 4_cell data used to draw this model's
sectors.

Below are the definitions for the sector, activity, boundary and chan_assign
data files.

================================================================================
================================================================================
Example of sector_data file generated by UTMOST with 4 sectors:

sector:         The sector number
  cell:         Calculate by (sector/3)+1
  csect:        Calculate by (sector%3)+1
secloc:         The direction at which this sector is located with respect
                to the center base station location.
centerX:        The x coordinate in km at which the base station for this
                cell is located.
centerY:        The y coordinate in km at which the base station for this
                cell is located.
mult:           The basestation radius multiplier for this sector:
                radius * mult = range
noise:          The receiver noise at the base station.
smooth:         The smoothing weight for sector k.
cpcity:         The hard capacity of a sector.
```

Eb/N0: The minimum Eb/N0 requirement at sector k.

sector k (K)	secloc degrees	centerX (5km radius)	centerY	mult	noise sigma	smooth wk	cpcity C	Eb/N0 Ek
0	150	4.33	5.0	3.0				
1	30	4.33	5.0	3.0				
2	270	4.33	5.0	3.0				
3	150	12.99	5.0	3.0				
4	30	12.99	5.0	3.0				
5	270	12.99	5.0	3.0				
6	150	21.65	5.0	3.0				
7	30	21.65	5.0	3.0				
8	270	21.65	5.0	3.0				
9	150	30.31	5.0	3.0				
10	30	30.31	5.0	3.0				
11	270	30.31	5.0	3.0				

Example of sector_data file generated by UTMOST with 4 sectors:
(definitions of columns same as 4_sector_data file)

sector k (K)	secloc degrees	centerX (5km radius)	centerY	mult	noise sigma	smooth wk	cpcity C	Eb/N0 Ek
0	150	8.66	5.0	3.0				
1	30	8.66	5.0	3.0				
2	270	8.66	5.0	3.0				
3	150	17.32	5.0	3.0				
4	30	17.32	5.0	3.0				
5	270	17.32	5.0	3.0				
6	150	4.33	12.5	3.0				
7	30	4.33	12.5	3.0				
8	270	4.33	12.5	3.0				
9	150	12.99	12.5	3.0				
10	30	12.99	12.5	3.0				
11	270	12.99	12.5	3.0				
12	150	21.65	12.5	3.0				
13	30	21.65	12.5	3.0				
14	270	21.65	12.5	3.0				
15	150	8.66	20.0	3.0				
16	30	8.66	20.0	3.0				
17	270	8.66	20.0	3.0				
18	150	17.32	20.0	3.0				
19	30	17.32	20.0	3.0				
20	270	17.32	20.0	3.0				

Example of model_data file used by RACEWIN

```
970712160945   : Simulation name (same as directory)
1          : Number of steps simulated so far
50         : Number of users
7 Cell             : Cell Pattern
Constant   : Model type (behavior)
Arriving   : Model type
15         : Sector Overlap
```

UTMOST Activity Files

step: The step at which the numbers were calculated in the model. (will be same as file name)
user: The user ID.
x: The current X coordinate for user X (km).
y: The current Y coordinate for user Y (km).
speed: The current speed of user (kph).
dir: The current direction at which user is moving (degrees). 999 for not moving.
time: The time stamp for current step in seconds.
xmit: The current transmit power of user (0 if phone is off)
calling: Is the user currently calling, roaming, or just driving? (C, R, D respectively. Note if D, xmit must = 0. If D, but speed = 0, then user isn't moving and doesn't have phone on.)
antennae: Is the sector whose antennae is currently servicing this user.
sector: Is the sector in which user currently is located. (Note that Out means the user is currently not located inside the model boundaries. Hopefully in more final versions of UTMOST there will be no users Out of model... they will be recycled back into the model.)
crossed: Indicates user has crossed 1 or more boundaries this step.
visible sectors: Sectors in model which can be seen by this user based on directional antennae.

step	user i	x (km)	y (km)	speed	dir (deg)	time (sec)	xmit (W)	calling	antennae	sector	crossed	visible sectors
7	0	23.47	16.65	20	37.15	70	0.03	R	5C	Out	N	1C 2C 3B 4B 5C 6B 7B
7	1	18.56	8.52	101	8.80	70	1.23	R	2C	2C	N	1B 2C 3B 4B 5A 6B 7B
7	2	18.07	12.06	144	268.95	70	0.51	D	5A	5A	N	1C 2C 3B 4B 5A 6B 7B
7	3	21.75	16.09	105	242.27	70	1.41	C	5C	5C	N	1C 2C 3B 4B 5C 6B 7B
7	4	21.00	4.12	111	191.38	70	0.69	D	2B	Out	N	1B 2B 3B 4B 5B 6B 7B
7	5	17.76	3.75	72	165.09	70	1.38	R	2B	2B	N	1B 2B 3B 4B 5A 6B 7B
7	6	21.02	18.86	117	273.54	70	1.93	C	7B	7B	N	1C 2C 3B 4C 5C 6B 7B

Channel Assignment Result Files

step: The step at which the following numbers were calculated in the model. (will be the same as filename)
user: The user ID.
old power: The transmit power of user at time of crossing
new power: The transmit power of user after channel reassignment
old antennas: Is the antennas (sector location) which was servicing user.
new antennas: Is the new antennas (sector location) which is servicing user.

step	user i	old power	new power	old antennae	new antennae
7	1	1.10	1.10	1 3C	1 3A
7	2	0.34	0.34	1 4C	1 4C
7	3	1.47	1.47	1 3B	1 3A
7	4	0.33	0.33	1 4C	1 7A
7	5	2.42	2.42	1 7A	1 4C
7	6	1.24	1.24	1 2B	1 2C
7	7	0.86	0.86	1 2B	1 2B
7	8	2.80	2.80	1 2A	2 1B 1A

The definitions of columns are not part of the actual data file.

12. Appendix 4: User's Guide

12.1 What's Needed?

Because RACEWIN is a Java application, it can run on any machine which has a Java Virtual Machine (JVM). It should be able to be executed on a PC running Windows 95 or Windows NT, a Macintosh, a UNIX machine or an OS/2 machine as long as the machine has the JVM installed. To learn more about Java, go to the Sun Microsystem's JavaSoft home page: http://java.sun.com/.

Java Developer's Kit (JDK) 1.0.2 is the minimum version on which RACEWIN will run. RACEWIN has also been tested on more recent versions including JDK 1.1.1.

The files needed for RACEWIN are displayed in Table 7.

Table 7. Files Required by UTMOST

File Name	Description
4_sector_data	The 4 cell pattern description file
7_sector_data	The 7 cell pattern description file
AboutBox.class	AboutBox Dialog byte code
AboutBox.java	AboutBox Dialog source code. This give information about the UTMOST tool.
ColorCanvas.class	The canvas on which color legend goes. See Legend.java for source
ColorRange.class	Color range object byte code.
ColorRange.java	Color range object source code. This is used by the Legend class.
Console.class	Console Dialog byte code
Console.java	Console Dialog source code
DebugLevel.class	DebugLevel byte code
DebugLevel.java	DebugLevel source - used for displaying debug messages to the console.
DrawnRectangle.class	DrawnRectangle byte code
DrawnRectangle.java	DrawnRectangle source which will draw a Rectangle. Code from Graphic Java Book
EtchBox.class	EtchBox byte code
EtchBox.java	EtchBox source which will create an etched box. This is used to draw an etched box around a component and place a title in the top center of the box. Code from Graphic Java Book
EtchedRectangle.class	Etched Rectangle byte code.
EtchedRectangle.java	Etched Rectangle source code. This creates a rectangle with etched edges. Code from Graphic Java Book
Etching.class	Etching byte code.

File Name	Description
Etching.java	Etching source code. Etching creates an etched line effect. Code from Graphic Java Book
FilesPanel.class	FilesPanel byte code.
FilesPanel.java	FilesPanel is a panel for changing files from which various things are read. Is a component of the PropertiesSetting class. Currently does nothing.
Format.class	Format byte code. Code from Core Java Book
Format.java	Format source code. Format allows printf type formatting of output data. Code from Core Java Book
ImagePanel.class	ImagePanel byte code.
ImagePanel.java	ImagePanel source code. Panel for setting variables used when loading background image.
Legend.class	Legend byte code.
Legend.java	Legend source code. Legend is the Power Legend which shows user's color - power relationship.
LineSegment.class	LineSegment byte code.
LineSegment.java	LineSegment source code. This is an object which represents a simple line segment.
LoadDialog.class	LoadDialog byte code
LoadDialog.java	LoadDialog source code. Used for loading simulation data.
MapArea.class	MapArea byte code.
MapArea.java	MapArea source code. The MapArea is the canvas on which the patterns and users are drawn.
MapPanel.class	MapPanel byte code.
MapPanel.java	MapPanel source code. MapPanel is the container class which contains the MapArea, the Legend and the 4 buttons for showing different things.
ModelSetup.class	ModelSetup byte code.
ModelSetup.java	ModelSetup source code. ModelSetup is the top portion of UTMOST where various selections may be made which define the current model.
POCAT.class	POCAT byte code.
POCAT.java	POCAT source code. Power control algorithm class.
PowCtrlPanel.class	PowCtrlPanel byte code.
PowCtrlPanel.java	PowCtrlPanel source code. A panel for setting variables used by the power control algorithm (POCAT).
PropertySettings.class	PropertySettings byte code.

File Name	Description
PropertySettings.java	PropertySettings source code. PropertySettings contains the 6 customizing panels, UserPanel, SectorPanel, VariablesPanel, ImagePanel, PowCtrlPanel and FilesPanel.
QuitBox.class	QuitBox byte code.
QuitBox.java	QuitBox Dialog source code. This gives the “Yes” or “No” option for quitting.
Sector.class	Sector byte code.
Sector.java	Sector source code. Sectors are the 1/3 part of a cell.
SectorInfo.class	SectorInfo byte code
SectorInfo.java	SectorInfo source code. Calculates and displays sector information
SectorsPanel.class	Sectors Panel byte code.
SectorsPanel.java	SectorsPanel is a panel for changing or creating on the fly the sector properties. It is a component of the PropertiesSetting class.
User.class	User byte code.
User.java	User source code. A user has information like speed and direction of movement.
UsersPanel.class	UsersPanel byte code.
UsersPanel.java	UsersPanel is a panel for changing various user properties. It is a component of the PropertiesSetting class. Currently does nothing.
UserStats.class	UserStats byte code.
UserStats.java	UserStats source code. UserStats is the bottom portion of UTMOST where a selected (clicked on) user’s data may be displayed. This is also where the Model Control is (<-,Go,-> Buttons).
Util.class	Util byte code.
Util.java	Utility source code. This is where primarily the algebraic and geometric common functions are located.
UTMOST2.class	UTMOST2 byte code.
UTMOST2.java	UTMOST2 source code is where the main() method is.
VariablesPanel.class	VariablesPanel byte code.
VariablesPanel.java	VariablesPanel is a panel for changing various variables. It is a component of the PropertiesSetting class. Currently does nothing.

To execute RACEWIN, you must be in the directory in which the RACEWIN class files are (~racewin/src) and you may need to set the Java CLASSPATH environment variable to include the current directory ("setenv CLASSPATH ." for Unix csh.) To run RACEWIN type "java UTMOST2". The window in Figure 27 should then appear.

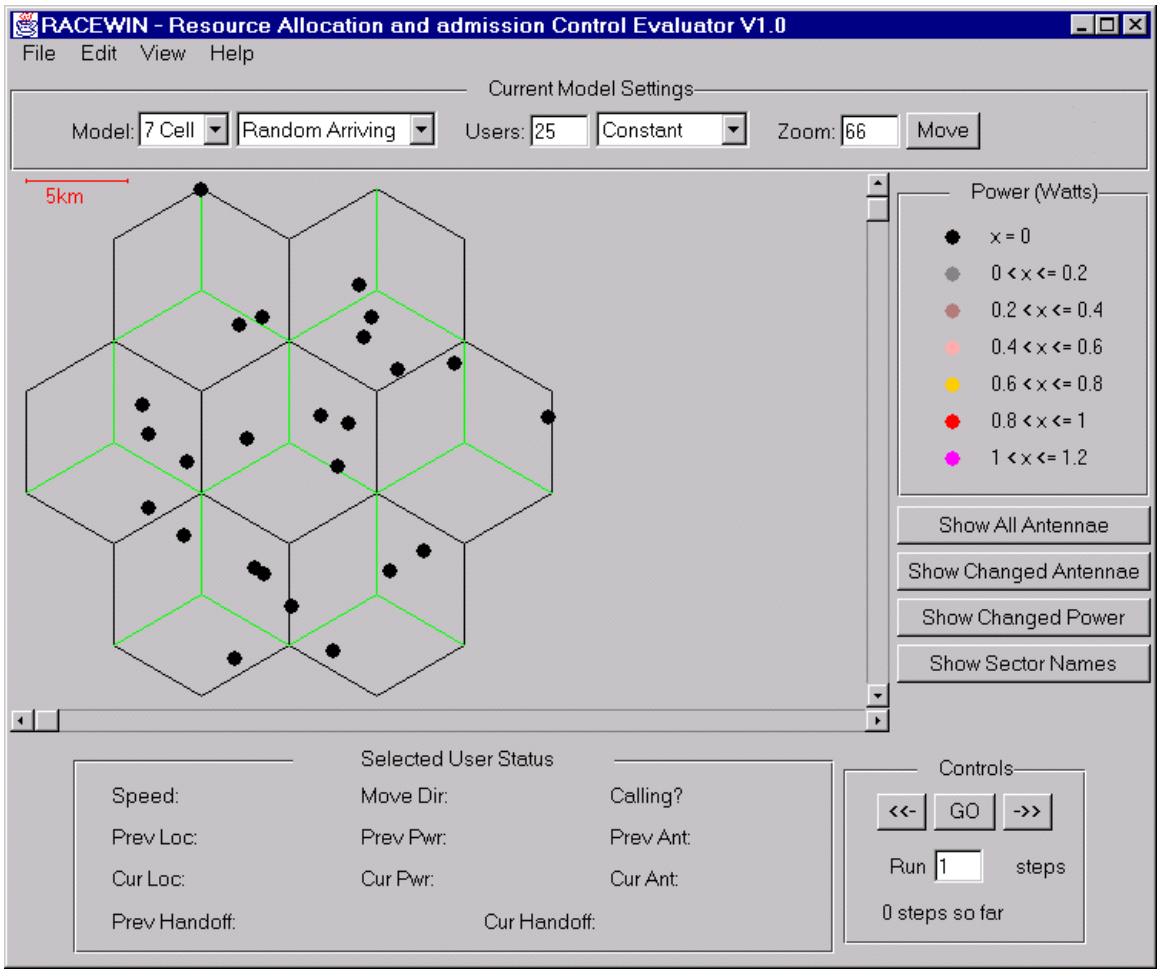


Figure 27. The RACEWIN GUI

12.2 Selecting Model Settings

Once RACEWIN is running, you will see the main UTMOST screen in Figure 27.

Next you may wish to select other initial model settings than the defaults shown. Select the cell pattern, model, and user behavior by using the "popup" menus shown in Figure 28 and type in the number of users you wish to have in the model.

Figure 28: Model Settings Panel

Constant behavior means that once a user begins its movements, if it moves outside the pattern or reaches its destination, it doesn't move. Regenerate means that the user will be regenerated with a completely new set of attributes when it moves outside the pattern or reaches its destination.

Table 8 shows the summary of all the pattern and model configurations which UTMOST currently supports.

Table 8. MOST Model Types

Pattern	Model Type	Description
7 Cell	Random Arriving	Users start from random locations inside the 7 Cell pattern and all move towards the center.
7 Cell	Edge Arriving	Users start from the edge (perimeter) of the 7 Cell pattern and all move towards the center.
7 Cell	Random Leaving	Users start from random locations inside the 7 Cell pattern and all move towards the outside edge of the model in a straight line from the center.
7 Cell	Center Leaving	Users start from the center of the model and all move towards the outside edge of the model in a straight line from the center.
7 Cell	Random	Users start from random locations inside the 7 Cell pattern and move in random straight line directions.
4 Cell	Random Arriving	Users start from random locations on the horizontal center line of the 4 Cell pattern and all move towards the right.
4 Cell	Arriving to Right	Users start from the far left edge of the 4 Cell pattern on the center line of the 4 Cell pattern and all move towards the right.
4 Cell	Random Leaving	Users start from random locations on the horizontal center line of the 4 Cell pattern and all move towards the left.
4 Cell	Arriving to Left	Users start from the far right edge of the 4 Cell pattern on the center line of the 4 Cell pattern and all move towards the left.
4 Cell	Random	Users start from random locations on the horizontal center line of the 4 Cell pattern and move in random left or right directions.

12.3 Properties Dialog

The Edit → Properties pull down menu option shows the properties dialog (PropertySettings object). This dialog is used to change the settings on some various variables. Table 9 shows all variables used and under which panel they are located.

Table 9: Properties Settings

Variables	***Description***
Cell Radius	Cell radius in km. Currently not implemented (hard coded to 5 km).
Time Increment	Time increment between each simulation step. Currently not implemented (hard coded to 10 sec).
User Dot Size	User Dot size. Number of pixels in dot radius
Max Power	The maximum Power (Watt) for color scale
Files	
Users	
Sectors	
Sector Overlap	Sector overlap angle (Degrees)
Number of Channels	Number of channels per sector (Maximum number of users that can be assigned)
Power Control	
Power Control	Selection of Power control type.
- No Power Control	New steps are created without calculating channel and power assignments.
- POCAT	RACEWIN's integrated version of POCAT is used to calculate channel and power assignments for each new simulation step.
- File Interaction	RACEWIN will wait for channel assignment files created by a stand alone tool
Fading Model	Selection of Fading model choice.
-Single Step	
-Multiple Step	Not implemented
Max Power*	The maximum Power that a user can be assigned with.
Min Power*	The minimum Power that a user can be assigned with.
Handoff Threshold*	
Noise	The noise power (Watt) received at each antennae
CIR	The Carrier to interference ratio
Maximum Iterations	The maximum number of iterations for the

	power control and channel assignment algorithm
Image	
Load Image (Checkbox)	Select to load image
Show Image	Name of image file to be displayed
Xoffset, Yoffset	

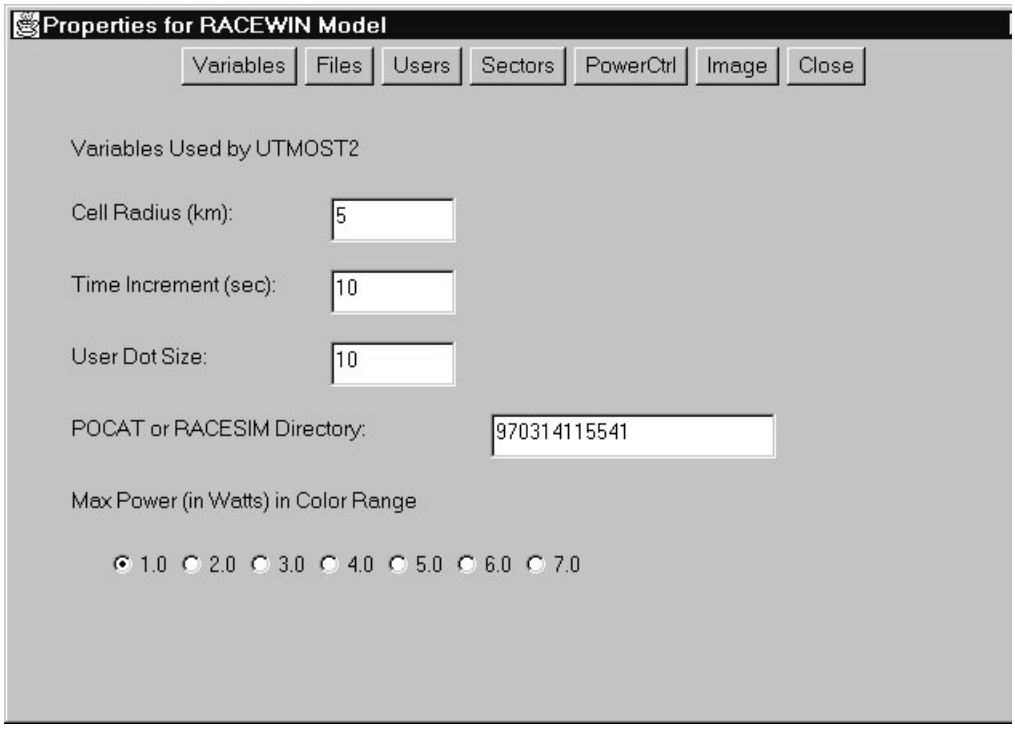


Figure 29. Properties Panel

Once the model has been defined, you may press "→" to run 1 or more steps. To run more than 1 step, change the "Run steps" number. You may also wish to display any of the following things by pressing the corresponding button.

Table 10. Show/Hide Buttons in UTMOST

Button Name	Description
Show/Hide All Antennae	The antennae assigned to each user is indicated by a black line from the user to the antennae.
Show/Hide Changed Antennae	After any step in which a user changes its assigned antennae, a green line will be shown from the user back to its previous assigned antennae and a red line will be shown from the user to its newly assigned antennae.
Show/Hide Changed Power	Any user whose power has changed will be shown with a red circle around the user dot.
Show/Hide Sector Names	The sector names will be shown in each sector. A name of 5B means the cell named 5 and sector named B.

12.4 Sector Information

During a simulation, information about the sectors in the network such as number of users and total received power can be displayed by selecting Sectorinfo in the View menu.

12.5 Loading Images and Moving the Network Pattern

To load an image in the background you need to enter the name of the GIF or JPG file in the properties window. Once an image is loaded, the network pattern can be moved over the image by first clicking on the move button in the upper left corner. When the network patters is in the desired position, click the “move” button again to “lock” the pattern.

12.6 Model Scenarios

The models available in UTMOST were described in the previous section. See Table 8 for the complete set of models. This section discusses briefly the intended usage of these different models.

The 7 Cell pattern is intended to represent mobile users arriving at or leaving an event like a football game. It may also be used as a random pattern. Generally, the random arriving and random leaving are not part of this football game model, but were included in UTMOST, because they may prove useful to represent a city at morning or evening rush hours.

The 4 Cell pattern is intended to represent mobile users moving along a highway. The random initial setups are probably most realistic for this kind of model. The users would start at points spreading out along the entire stretch of highway. They may all be arriving or leaving at rush hours, or they may just randomly traverse in either direction. Similar to the 7 Cell model, there are patterns which may or may not be useful. These are arriving and leaving patterns, where all users start on the far left or the far right, respectively.