GLOBAL OPTIMIZATION IN THE DESIGN OF MOBILE COMMUNICATION SYSTEMS

A Thesis

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Master of Science

by

Bhaskar Krishnamachari

May 1999

Abstract

Intensive research has been focused in recent years on the development of sophisticated mobile communications systems with increased network capacity and performance that utilize scarce resources in an optimal manner. A number of design and resource allocation problems in mobile communications are known to be NP-hard combinatorial optimization problems characterized by extremely large search spaces. Such problems are intractable to solution using simple analytical or deterministic approaches. Search techniques for global optimization such as Random Walk, Simulated Annealing, Tabu Search and Genetic Algorithms, on the other hand, have been found to provide near-optimal solutions in reasonable time for such problems.

This thesis describes the results of a research program that focused on the relative performance of these search algorithms for two problems encountered in the design of mobile communications – the design of a low-cost topology for the fixed portion of a cellular network, and the optimal location of base stations to maximize radio coverage while minimizing equipment costs. The performance of each technique is examined for various algorithm and problem parameters. The experimental results yield some insights into the relative suitability of these techniques for each problem. Both intuitive and analytic explanations are provided for any significant differences in performance.

Biographical Sketch

Bhaskar Krishnamachari was born in 1977 in Nagpur, India. Till the age of 15, he spent most of his life in New Delhi, India, where he attended the Naval Public School. In 1994, one year after his arrival in the United States, he graduated from Humanities high school in New York. He attended the prestigious Cooper Union for the Advancement of Science and Art in New York City under a four year full tuition scholarship, and he graduated *summa cum laude* as a Bachelor of Engineering in Electrical Engineering in 1998. During the course of his undergraduate education he received several awards and honors including the 1998 IEEE Region 1 best student paper award, the All American Scholar Collegiate Award, and the National Dean's List. He is currently in his first year at Cornell University pursuing an M.S./Ph.D. in Electrical Engineering under a three-year Cornell Graduate Fellowship. He has also been selected as one of eight 1999 Olin Presidential Fellows at Cornell University. He has presented three conference papers to date, and currently has a U.S. patent pending for "Non-Semiconductor Passive Microwave Logic Devices." Bhaskar is a member of the Tau Beta Pi engineering honor society, the Eta Kappa Nu honor society for electrical engineers, and the Order of the Engineer. He is also a member of the Institute for Electrical and Electronic Engineers (IEEE), and the Association for Computing Machinery

(ACM). His non-academic interests include cinema, poetry, fiction, history, languages (besides English, he knows Hindi, Tamil, French, and a bit of Mandarin Chinese), tennis, and dreaming .

To Amma and Appa

Acknowledgements

No thesis can be the result of a student's solitary activities. I would like to take this opportunity to acknowledge the others whose wisdom and ready assistance I have relied upon in some form or another. Thanks, first of all, to Dr. Stephen Wicker, the chairman of my advisory committee, for his encouragement, support and guidance without which I could not have so easily developed and tested my ideas; and to Dr. Bart Selman, my minor advisor, for his constructive comments and useful pointers that helped me greatly in my work. Thanks also to my friend and colleague Xi Xie (Sid) for helping me feel right at home here at Cornell, and for some useful discussions on the nature of search for global optimization. The Markov framework for analyzing the performance of search techniques mentioned in chapter 4 and the corresponding figures demonstrating the performance of Random Walk algorithms on a small search space were developed jointly with him. Thanks to the Olin Foundation and the Graduate School of Cornell University for their generous financial and academic support.

I would also like to acknowledge the founder of my undergraduate institution – the late Peter Cooper, a selfless, generous man whose dream that education ought to be "as free as air and water" has inspired all who benefited from it. Thanks to my colleagues at Cooper Union: Sajan Abraham, Christopher Gracia, Simon Lok, John Davis, Gabriel Loh, Stephen Lu, Kevin Chou, Walter Lin, Andrew Suwidji, and Irene Bodonenko, who really gave me confidence in my abilities and stimulated my desire to pursue my graduate studies in Electrical Engineering. My grateful thanks to Zhen Nie for the constant support and for helping me keep things in proper perspective.

I would like to thank all the teachers, and professors I have had during my school and college years that served as role models, gave me needed advice, encouraged and motivated me to pursue my educational goals, taught me to see the world in interesting new ways, and showed me what it takes to achieve one's aims. Some that come directly to mind are Mrs. Vasantha, Mr. Vashisht, and Mrs. Aisola from NPS, New Delhi; Mrs. Zelicoff, Mr. Downes and Mr. Kaplan from Humanities H.S., New York; Prof. Czarkowski from Polytechnic University, New York; Chuck Kim, Prof. Stewart, Prof. Kaplan, Prof. Brazinsky, Prof. Ungar, Prof. Kirtman, Prof. Cumberbatch, and Prof. Sandler from Cooper Union, New York.

Finally, I thank my parents T. R. Krishnamachari and Vijayalakshmi Krishnamachari for all their love and encouragement. They have been a great source of inspiration and support for me. This thesis is dedicated to them.

Table of Contents

List of Figures

Chapter 1

Introduction

1.1 Personal Communication Systems

In the last two decades, wireless communication systems such as cordless phones, paging systems, wireless data networks, satellite-based and cellular mobile systems have been gaining steadily, both in popular importance and technological sophistication. The first-generation wireless systems were developed in the late 1970's and 1980's and were based on analog technology (such as the Advance Mobile Phone Service (AMPS) by AT&T and Nordic Mobile Telephone (NMT) by Ericsson). As demand increased and digital technology matured in the 1980's and 1990's the second-generation digital wireless systems were designed (such as Global System for Mobile Communications (GSM) in Europe and Digital AMPS in North America). These systems, currently in use, offer higher system capacity and improved quality of service. Third-generation systems, referred to as personal communication systems (PCS), are currently under development and are expected to be deployed at the beginning of the 21st century [1], [2].

The utilization of wireless communications has shown growth rates of 20-50% per year in various parts of the world. As the benefits of digital technology are realized, it is expected that there will be demand for the transmission of high-bandwidth, high-quality multimedia information content over these systems. The PCS goal is to provide to every user the ability to exchange such information securely with anyone, anytime, anywhere in the world, using a unique personal telecommunication number (PTN). To meet these challenging expectations, intensive research has been undertaken in recent years to develop a sophisticated PCS with increased network capacity and performance. One of the trends in this research has been the growing incorporation of Artificial Intelligence (AI) techniques into such systems [3].

Like all engineering endeavors, the subject of mobile communications also brings with it a whole host of complex design issues. A number of issues in resource allocation, design and planning of mobile communications can be formulated as combinatorial optimization problems. Many of these problems are NP-hard [4], characterized by search spaces that increase exponentially with the size of the input. They are therefore intractable to solution using analytical approaches or simple deterministic algorithms. Heuristic and stochastic search based optimization procedures from AI offer more appropriate alternatives.

1.2 Search Techniques for Global Optimization

At about the same time as the advent of the first generation wireless systems, some robust and effective search techniques for optimization were invented - Genetic Algorithms (GA) (Holland, 1975 [5]), Simulated Annealing (SA) (Kirkpatrick et al., 1983 [6]), and Tabu Search (TS) (Glover, 1986 [15]). These techniques have proved to be very effective in providing good solutions to NP-hard combinatorial optimization problems in polynomial time[17]. The application of these techniques to a large number of fields such as industrial production, management, financial services, game theory, telecommunications, graph theory, biological modeling, and VLSI has been increasing steadily from the mid 80's through the 90's.

The application of these search algorithms to the field of mobile communications is still in its infancy. As of the date of writing, there have been little more than twenty papers published describing the use of these techniques for problems in mobile communications, nearly all written in the past five years. This is certain to change - there is a growing interest in the utilization of these techniques to find better, more optimal solutions to problems where ad-hoc approaches have been utilized hitherto due to computational intractability.

1.3 Thesis Description and Outline

This thesis is concerned with the empirical analysis of the performance of search techniques for two optimization problems that arise in the planning of mobile communication systems - the design of a low-cost topology for the fixed portion of a cellular network, and the location of base transmitting stations to maximize radio coverage while minimizing equipment and maintenance costs. Chapter 2 provides a summary of combinatorial optimization, and describes the four search algorithms for global optimization that are analyzed in this thesis - Random Walk, Simulated Annealing, Genetic Algorithms, and Tabu Search. Chapter 3 demonstrates the difficulties inherent in attempting to describe the performance of these algorithms analytically, and provides the rationale for empirical analysis of these algorithms. Some necessary considerations of fairness while performing experimental comparison of algorithms are also described. A survey of the existing literature on the application of search techniques to problems in mobile communications is provided in Chapter 4. Chapter 5 describes the base station location problem and presents the results of experiments that studied the performance of each search techniques for different problem and algorithm parameters. A head-to-head comparison of the various algorithms was also performed for this problem. Chapter 6 describes the network topology optimization problem, and presents similar experimental data showing the performance of the search techniques for different parameters. Chapter 7 provides some concluding comments.

Chapter 2

Search Techniques for Global Optimization

2.1 Combinatorial Optimization

An optimization procedure seeks to find the optimal solution $x^* = min\{f(x)|x \in S\}$, where f is the cost function (also known as the objective function) with domain S - the set of possible solutions to the given problem. A combinatorial optimization problem is defined as an optimization problem where S has discrete members. When the problem complexity is low, either an exhaustive search of the space, or deterministic algorithms (such as Linear and Nonlinear Programming [50], Dynamic Programming [51], Branch-and-Bound algorithms and Polyhedral Cutting Plane approaches [52]) may be used to obtain the solution. For more difficult problems, heuristic and stochastic search techniques must be employed to find the optimal point in a large solution space.

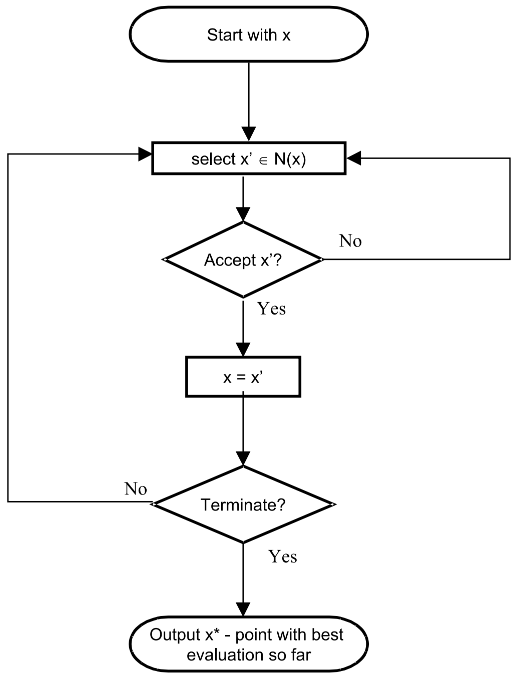


Figure 2.1: A General Neighborhood-based Search Algorithm for Optimization

2.2 Neighborhood Search

A subset of S, designated $N(x)$, may be associated with each point $x \in S$. $N(x)$ is referred to as the *neighborhood* of x. At each step, most search techniques for optimization proceed from the current point x by considering a point or a set of points in $N(x)$. Such neighborhood-based search algorithms are then differentiated from each other by the way in which the considered points are accepted (via some selection criterion based on cost function evaluations). The common ingredients of a neighborhood based search technique are shown in Figure 2.1. Local search techniques such as steepest descent algorithms move from point to neighboring point, accepting each successive point as a solution only if it has a lower cost than the current solution. This causes entrapment in the point with the lowest cost function in the neighborhood - a local optimum. Global search techniques[1] provide an escape from such traps by providing the ability to selectively accept successive points even if they have a higher cost than the current solution. This chapter describes the four algorithms examined in this thesis: Random Walk (RW), Genetic Algorithms (GA), Simulated Annealing (SA), and Tabu Search (TS). These algorithms are very simple and easy to implement (perhaps the chief reason for their popularity), and are very general and robust in their simplest form - assuming nothing about the structure of the space being searched. To improve their efficiency when applied to specific problems, they may be modified suitably (via better neighborhood definitions) or even hybridized with heuristic techniques. Only a brief description is provided in this chapter; for an excellent introduction to the subject of search techniques for combinatorial optimization see [17].

2.3 Random Walk

Random Walk is perhaps the simplest of the neighborhood-based search techniques. At each step, a point $x' \in N(x)$ is generated from the current point x. If x' has a cost function evaluation not greater than the current point ($f(x') \leq f(x)$), it is accepted and the search proceeds by setting $x = x'$. If $f(x') > f(x)$, however, the point x' is accepted with a probability p - which is the probability of accepting uphill moves. This is shown in Figure 2.2. Varying the parameter p from 0 to 1,

[1]There is some confusion regarding nomenclature in the literature. These techniques are sometimes referred to as "local" search techniques (e.g. [17]) because they proceed by searching neighborhoods in the search space. In this chapter, we distinguish between local search procedures (such as steepest descent algorithms) that are susceptible to being trapped in local optima, and global search procedures such as Genetic Algorithms, Simulated Annealing and Tabu Search, that are capable of escaping such minima and providing globally optimal solutions.

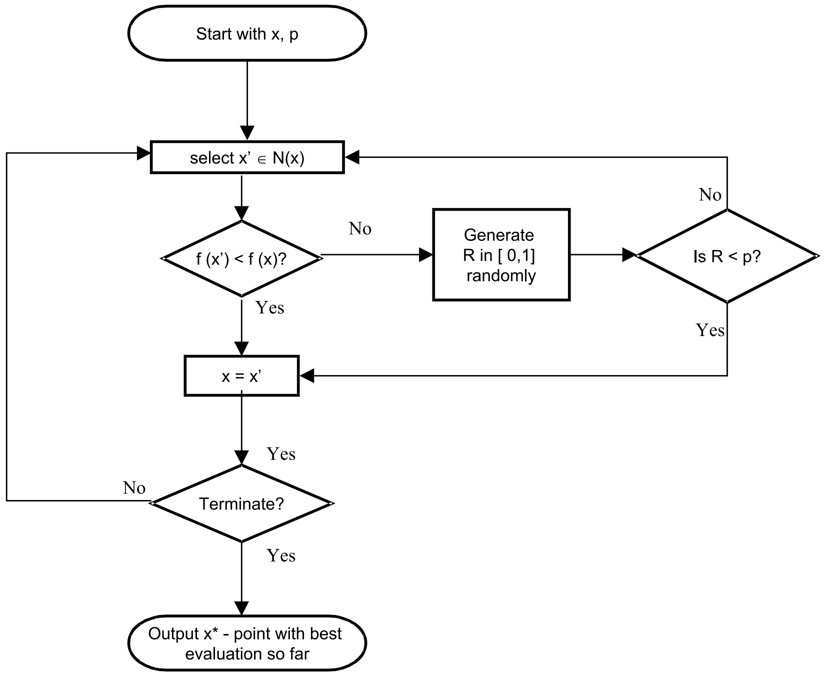


Figure 2.2: Random Walk

we get a whole family of Random Walk algorithms that range from greedy search ($p = 0$) to purely random search ($p = 1$). In several problems of interest one may expect that since greedy search will result in the search converging to a local minimum, a small non-zero uphill probability can help in escaping such minima. When the uphill probability is too high, however, the search becomes more random and there may be a performance drop due to this. For some problems, a greedy search that is repeated from different starting points (sometimes known as multiple-start local search (MLS)) may be utilized to increase the possibility of locating near-optimal solutions.

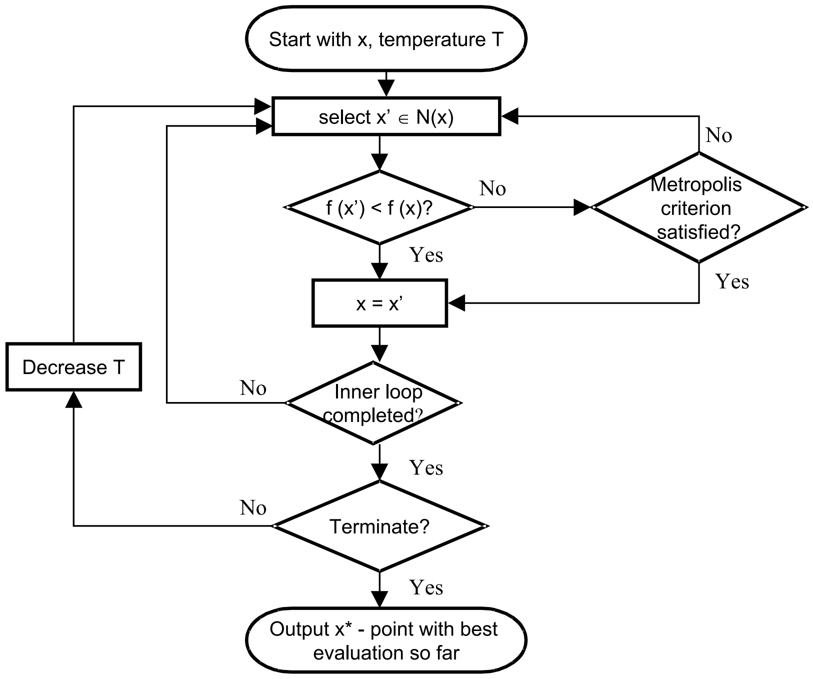

Start with x, temperature T
select x' ∈ N(x)
f (x') < f (x)?
No
Metropolis criterion satisfied?
No
Yes
Yes
x = x'
Decrease T
No
Inner loop completed?
Yes
No
Terminate?
Yes
Output x* - point with best evaluation so far

Figure 2.3: Simulated Annealing

2.4 Simulated Annealing

This algorithm derives its inspiration from the thermodynamic process by which solids are heated and cooled gradually (annealed) to a crystalline state with minimum energy. At each step in the algorithm a point $x' \in N(x)$ is generated from the current point x. If the point has a lower cost function than x it is accepted unconditionally, but even if it has a higher cost it is accepted probabilistically using the Metropolis criterion described below. This acceptance probability is proportional to the temperature T of the annealing process, which is lowered gradually as the algorithm proceeds. The procedure is illustrated in Figure 2.3.

The Metropolis criterion is as follows: for $x' \in N(x)$, the probability that x' is selected is

$$P_{x \to x'} = min\left(1, \exp\left[-\frac{f(x') - f(x)}{T}\right]\right) \tag{2.1}$$

When T is high initially, there is a greater probability of making uphill moves, which allows the search to fully explore the space. It has been shown, by modeling SA as Markov processes [12], that Simulated Annealing will converge asymptotically to the global optimum under two conditions:

- Homogeneous condition: if the temperature is lowered in any way to 0, while the length of the homogeneous Markov sequence (formed by the accepted points) at each temperature is increased to infinite length.
- Inhomogeneous condition: if irrespective of the length of these isothermal Markov chains, the cooling schedule is chosen such that T approaches 0 at a logarithmically slow rate.

Since in practice neither of these is possible in finite implementations, polynomial time approximations are used. The choice of cooling schedule and the length of the Markov chains at each temperature affects the quality of the results and the rate of convergence. The definition of the neighborhood function (which is usually based on some heuristic understanding of the problem at hand) determines how new points are visited. The SA program is terminated if an acceptable solution is found or if a designated final temperature is reached.

Simulated Annealing has quite a dedicated following and has been very successful in a broad range of NP-hard optimization problems. Quite a few papers have been written to date demonstrating the application of SA to such problems in mobile communications.

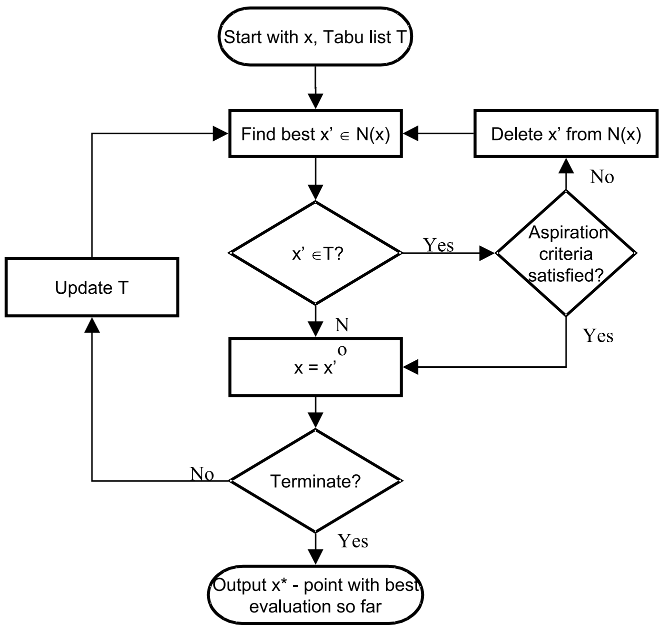

Start with x, Tabu list T
Find best x' ∈ N(x)
Delete x' from N(x)
No
x' ∈T?
Yes
Aspiration criteria satisfied?
Update T
N o
Yes
x = x'
No
Terminate?
Yes
Output x* - point with best evaluation so far

Figure 2.4: Tabu Search

2.5 Tabu Search

Tabu Search is based on the premise that intelligent problem solving requires incorporation of adaptive memory [13], [14], [15]. Tabu Search is also a global search technique in that it provides means for escaping local minima. Figure 2.4 illustrates how TS works.

In TS, a finite list of forbidden moves called the tabu list T is maintained. At any given iteration, if the current solution is x, its neighborhood $N(x)$ is searched aggressively to yield the point x' which is the best neighbor such that it is not on the tabu list. Note that it is not required that $f(x') \leq f(x)$, only that $f(x') = min(f(x^+)|x^+ \in N(x) \setminus T)$. Very often though, to reduce complexity, instead of searching all the points in N(x), a subset of these points called the *candidate* *list* is considered at each step. The size of the candidate list may even be varied as the search proceeds. As each new solution x' is generated, it is added to the tabu list and the oldest member of the tabu list is removed. Thus the tabu list prevents cycling by disallowing repetition of moves within a finite number of steps (determined by the size of the list). This, along with the acceptance of higher cost moves, prevents entrapment in local minima. It may also be desirable to include in the tabu list attributes of moves rather than the points themselves. Each entry in the list may thus stand for a whole set of points sharing the attribute. In this case, it is possible to allow certain solutions to be acceptable even if they are in the tabu list by using what are called *aspiration criteria.* For example, one such criterion is satisfied if the point has a cost that is lower than the current lowest cost evaluation. If a neighborhood is exhausted, or if the generated solutions are not acceptable, it is possible to incorporate into the search the ability to jump to a different part of the search space (this is referred to as *diversification*). One may also include the ability to focus the search on solutions which share certain desirable characteristic (*intensification*) by performing some sort of pattern recognition on the points that have shown low function evaluations in the past.

Tabu search is a meta-heuristic technique, and it must be adapted to the problem at hand for it to be efficient. The choice of moves that generate the neighborhood of a point is problem-specific. Different implementations can be generated by varying the definition and structure of the tabu list (for example by deciding how tabu attributes are determined), the aspiration criteria, the size of the candidate list, intensification and diversification procedures etc. To speed up the search, faster ways to determine the best neighbor are required.

TS has also been applied successfully to a large number of NP-hard optimization problems and has been shown to compare favorably with GA and SA [17]. A search of the literature, however, has not revealed extensive application of TS in mobile communications though there is certainly no *a priori* reason why it cannot be applied to the same problems as GA and SA.

2.6 Genetic Algorithms

These algorithms derive their inspiration from the natural process of biological evolution. Solutions are encoded (often in binary) into strings or chromosomes. The algorithm operates on a population of these chromosomes, which evolve to the required solution through operations of fitness-based selection, reproduction with crossover and mutation that are fundamentally similar to their natural analogs. Figure 2.5 shows a diagram for the simple Genetic Algorithm. Using Markov chain modeling, it has been shown that GA's are guaranteed to converge asymptotically to the global optimum if an elitist strategy is used where the best chromosome at each generation is always maintained in the population [8]. More detailed descriptions of these algorithms, their implementation and applications can be found in [9], [10], [11].

Briefly, the process is as follows: an initial population P of N chromosomes is chosen. At each generation, a new population P' is generated by repeating the following procedure: first, two parents are picked from P in such a way that the probability of selection is proportional to their fitness. Then, new offspring are generated from these parents by crossover with some probability P_c. Finally, with

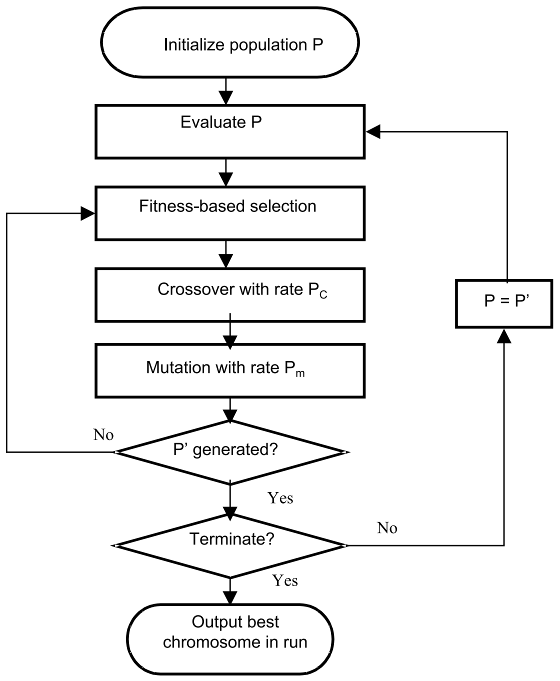


Figure 2.5: Genetic Algorithm

a probability P_m (often chosen to be very low: 0.001 - 0.01), these recombined chromosomes are then randomly mutated (for binary GA this refers to complementing each bit with probability P_m). The population thus generated replaces the current population. The procedure is repeated until some termination criterion is satisfied. Termination criteria may be defined such that the algorithm stops when an acceptable solution is found in the population or after a predefined number of function evaluations or generations have been processed.

Very often, for specific applications, the crossover and mutation operators may be redefined or extended, and a local search may be incorporated at the end of each generation to speed up convergence. The encoding of the solution space into chromosomes, the size of the population, and the mutation and crossover probabilities all vary widely depending on the implementation.

The traditional mutation and crossover operators do in some sense define a neighborhood for each chromosome in a population, but it is possible to relate Genetic Algorithms to the other neighborhood search schemes in a more direct way if we do not consider crossover. This is done by specifying that the mutation operator for the chromosome corresponding to point x in the search space is such that it yields a chromosome corresponding to point x' in the search space with the property that $x' \in N(x)$. This is the approach considered in this thesis. It is noted here that the significant difference between GA and the other search techniques described here is that it allows for a more parallelized search of the space since a population of points is considered at each step instead of just a single point.

Genetic Algorithms have been quite successful in a variety of applications and enjoy massive popularity, at least amongst members of the Evolutionary Computation community. For optimization problems in telecommunications, applications of Genetic Algorithms are by far the most numerous. The name *evolutionary telecommunications* has recently been suggested for this growing field.

Chapter 3

Applications of Search Techniques to Mobile Communications

In this chapter, papers describing the applications of these global search techniques to optimization problems in mobile communications are reviewed. The number of problems in mobile communications that have been formulated and recognized as hard combinatorial optimization problems and processed with these algorithms is as yet quite small. Thus, while the papers discussed here by no means cover the entire range of possible applications of these techniques to mobile communications, they do represent nearly all such attempts to date, and should provide a good overview of the subject.

It will be noted that most of the papers surveyed here are concerned with the application of Genetic Algorithms. This is not the result of any inherent superiority in terms of efficiency or ease of programming in using GA's as opposed to SA and TS, but perhaps an indication of its general popularity among researchers interested in optimization for wireless telecommunications.

Several papers discussing global optimization for system-level resource allocation, design, and planning in cellular mobile systems are surveyed here. Papers covering the use of these techniques for lower-level problems such as optimal multi-user detection in CDMA technology, design of frames in TDMA schemes, blind channel identification and equalization are also reviewed.

3.1 Fixed Network Topology Design

In cellular communication networks, the mobile units are connected to the public networks, such as PSTN, ISDN and other data networks, through a network hierarchy determined by the system architecture. Figure 3.1 shows such an architecture for the European GSM standard [16]. The mobile stations (MS) in each cell communicate over the radio interface with the Base Transceiver Stations (BTS). BTSs connect to Base Station Controllers (BSC) via microwave links or dedicated leased lines on what is called the *Abis interface*. Each BSC may control several hundred BTSs, and some low-level functions such as mobile handoffs are made by the BSC. The BSCs in turn are connected to the Mobile Switching Centers (MSC) via the *A interface*. The MSC controls traffic among all the BSCs and there is a sub-network between MSCs at this level that includes units such as the Home Location Register (HLR), Visitor Location Register (VLR), the Authentication Center (AuC), and the Point of Interconnect (POI) to the public networks. Thus there is a large part of the communication hierarchy in a mobile cellular system which is a fixed wired network.

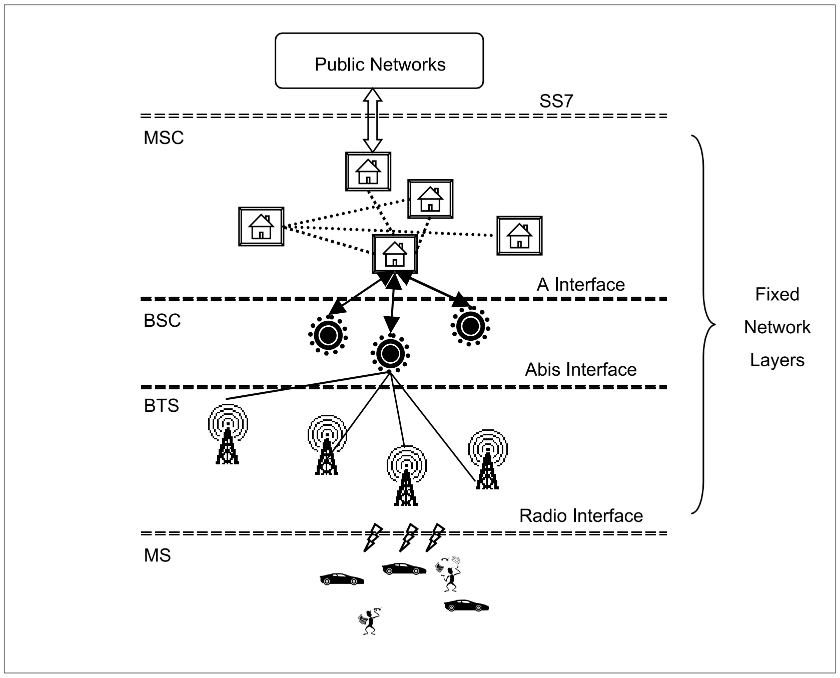


Figure 3.1: GSM Network Architecture

The cost of the topology of a fixed network depends upon several factors, including the cost of the nodes, the cost of links, link flow and capacity, and constraints such as the maximum number of links for each node. The problem of designing minimum cost network topologies is closely related to complex problems in graph theory; an exhaustive enumeration of topologies to obtain the optimal arrangement becomes infeasible for even moderately sized networks. Network topology design is perhaps the most studied application of global search techniques to telecommunications (for example [18],[19], [20], [21]).

The design of the fixed portion of the GSM network and the closely related Digital Cellular System 1800 (DCS1800) is performed in [22] using a GA-based Genetic Optimizer for Topological Network Design (GOTND). The cost function to be minimized is defined as follows:

$$f = \sum_{\forall n} C_n^{NODE} + \sum_{\forall p} C_p^{POI} + \sum_{\forall l \in L^{BTS \to BSC}, L^{BSC \to MSC}, L^{MSC \to MSC}} C_l^{LINK} \tag{3.1}$$

subject to (i) $F_l \leq C_l, \forall l$ and (ii) $\xi \leq 0.001$, where C_n^{NODE} is the cost of all nodes of type n, C_p^{POI} is the cost of the p^{th} POI, and C_l^{LINK} the cost of the l^{th} link which is one of types $L^{BTS \to BSC}$, $L^{BSC \to MSC}$, or $L^{MSC \to MSC}$; F_l, C_l represent the flow and capacity of each link; and ξ is the call blocking probability.

In this paper each candidate solution is represented by a set of seven chromosomes. The first four chromosomes represent the x and y coordinates for the BSCs and MSCs. The last three chromosomes describe the existence of links between BTSs and BSCs, between BSCs and MSCs and between MSCs themselves. The standard fitness based selection mechanism is used. Instead of the generic crossover and mutation operations, however, the GOTND employs 17 problem-specific operators. These include "Move one/some BSCs randomly", "move one/some MSCs randomly", "move MSC and BSC coupled randomly", "move MSC to BSC", "push BSC toward all connected BTS(s)", "push BSC(s) toward connected MSC" etc. among others. Different weighted combinations of these operators are tested to determine those that yield a fast convergence rate. Thus the operators that are more successful are used more often than less successful operators during the optimization.

In [22] the GOTND is applied to an example scenario with 100 BTSs, 4 BSCs and one MSC that is also the POI to the public telecommunication networks. The cost data for the links is assumed to be piecewise linear and was obtained from German Telecom. The results showed that a cost reduction of 19% over the best network in the first iteration could be achieved using the GOTND.

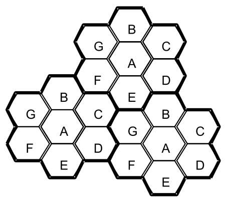


Figure 3.2: Frequency Reuse in 7-cell Clusters

3.2 Channel Allocation

One of the most basic limitations of mobile radio communication systems is the restricted spectral bandwidth. Fundamental to the cellular concept is the idea of frequency reuse by which the same frequencies/channels can be reused in different geographical locations [2], [16]. Each such location is referred to as a cell (usually hexagonal in shape), and is allocated a set of channels according to the expected demand in that cell. The entire spectrum is allocated to a cluster of cells arranged in shapes that allow for uniform reuse patterns. The geometry of the hexagonal cells restricts the number of cells per sector N to the discrete values

$$N = i^2 + ij + j^2 \tag{3.2}$$

where i and j are integers. Thus clusters can only accommodate 1, 3, 4, 7 cells. Figure 3.2 shows an illustration of the frequency reuse in a 7-cell cluster system.

The allocation of the limited number of channels to each cell in a mobile communication system is a much-studied problem [25]. The channels must be allocated in such a way as to satisfy

- Co-channel interference constraints : These constraints are due to radio interference between cells that have the same channels,

- Co-cell (or adjacent channel) constraints : These constraints arise because of the radio interference between adjacent channels in the same cell that are not separated by some minimum spectral distance, and
- Cell demand requirements: These indicate how many channels are required in each cell due to their unique traffic patterns.

It has been shown that the channel allocation problem (CAP) is equivalent to the generalized coloring problem in graph theory, a problem that is known to be NP-complete [36]. There are essentially two kinds of allocation schemes - Fixed Channel Allocation (FCA) and Dynamic Channel Allocation (DCA). In FCA the channels are permanently allocated to each cell, while in DCA the channels are allocated dynamically upon request. DCA is desirable, but under heavy traffic load conditions, FCA outperforms most known DCA schemes [26]. Since heavy traffic conditions are expected in future generations of cellular networks, efficient FCA schemes become more important. All of the papers surveyed that apply global search techniques to CAP are FCA schemes [27],[28], [29],[30], [31], [32].

Let us assume that there are C channels to be assigned to N cells. Typically, the interference constraints are modeled by an N-by-N compatibility matrix D. The diagonal elements of this matrix, $D_{x,x}$, represent the co-cell constraint - the number of frequency bands by which adjacent channels assigned to cell i must be separated . The non-diagonal elements $D_{x,y}$ represent the number of frequency bands by which channels assigned to cells x and y must differ. If the compatibility matrix is binary, then these constraints are expressed more simply - if the same channel cannot be reused by cells x and y then $D_{x,y} = 1$, and if it can be reused then $D_{x,y} = 0$.

The traffic requirements for the cells are modeled by a demand vector T of length N that represents the number of required channels in each of the cells. The assignment to be generated is denoted by an N x C binary matrix A which is chosen such that

$$A_{x,k} = \begin{cases} 1 & : \; if\ cell\ x\ is\ assigned\ channel\ c_k \\ 0 & : \; otherwise \end{cases} \tag{3.3}$$

In general, the cost due to the violation of interference constraints in this problem is given as:

$$f' = f_{co-cell} + f_{cochannel} = \alpha_1 \sum_{x}^{N} \sum_{i \neq j}^{C_x} \sum_{j}^{C_y} \Phi(i,j) + \alpha_2 \sum_{x \neq y}^{N} \sum_{y}^{N} \sum_{i}^{C_x} \sum_{j}^{C_y} \Psi(x_i, y_j) \tag{3.4}$$

where $\Phi(i,j) = \begin{cases} 0 & : \; iff\ |c_i - c_j| < D_{x,x} \\ 1 & : \; otherwise \end{cases}$ is a measure of the co-cell constraint satisfaction, and $\Psi(x_i, y_j) = \begin{cases} 0 & : \; iff\ |x_i - x_j| < D_{x,y} \\ 1 & : \; otherwise \end{cases}$ is a measure of the co-channel constraint satisfaction, x_i and y_j the assigned frequencies for the i^{th} and j^{th} channels of cells x and y respectively , C_x the number of channels in the x^{th} cell, and α_1, α_2 are constants to weigh the relative importance of the two constraints. The cost due to the violation of traffic demand requirements can be modeled explicitly as an error term:

$$f_{traffic} = \sum_{x}^{N} (T_x - \sum_{k} A_{x,k})^2 \tag{3.5}$$

The cost function to be minimized can be expressed as:

$$f = f' + f_{traffic} \tag{3.6}$$

If the traffic demand requirements are incorporated implicitly by only considering those assignments which satisfy them, then the cost function only consists of the interference-constraint violation term:

$$f = f' = f_{co-cell} + f_{cochannel} \tag{3.7}$$

subject to $\sum_{k} A_{x,k} = T_x, \forall x$. Some of the papers that describe the application of global search techniques to this problem are described below:

i) In [27], a Genetic Algorithm is used to determine an optimal channel assignment. In the encoding chosen in this paper for the GA, chromosomes whose total length is the sum of all channels required for each cell represent each possible allocation. The traffic demand is therefore incorporated into the representation. A typical chromosome consists of a linear arrangement of the channels for each cell listed in order. The standard mutation operator is chosen, while a slightly modified partially matched crossover (PMX) operator is designed that performs a crossover while resolving (correcting) any channel constraint-violations that may arise from the crossover to improve the performance of the algorithm. The algorithm was tested on a homogeneous cellular network of 49 cells where only three channels are available, and on data taken from an actual inhomogeneous cellular network consisting of 25 cells and 73 channels (known to be sufficient). In both cases the algorithm was able to generate conflict-free allocations. In the case of the inhomogeneous network example, the setting of a higher value of α_2(i.e. emphasizing co-channel constraint satisfaction over co-cell constraint satisfaction) was empirically observed to speed up convergence.

ii) Genetic Algorithms are also used in [28] for frequency planning. A binary compatibility matrix is used to model the co-channel constraints. The paper presents the performance of two versions of genetic algorithms - the simple GA with standard operators and binary representation, and a hybrid GA where an integer representation is used for the channels. For the simple GA, the demand requirements are explicitly incorporated into the cost function as in Equation (3.6). For the hybrid GA a representation which has the demand vector built into it is chosen. Thus a modified fitness function is chosen consisting only of the co-channel constraint satisfaction term. The hybrid GA uses uniform crossover instead of 1-point crossover (referring to the number of loci where recombination takes place). Furthermore, a steepest descent (local search) algorithm is incorporated at the end of each generation to perform a more thorough search of the solution space. The neighborhood for this local search is defined by selecting a different channel for each cell in conflict in a particular assignment and choosing the best such allocation.

The two algorithms were tested on cellular network scenarios with uniform and non-uniform traffic loads for two kinds of cellular systems - the standard planar system (with 12 channels and 21 cells) and a linear system (12 channels and 9 cells laid out side-by-side). It was observed empirically that the hybrid GA performs better than the simple GA, which, for a fixed upper limit on number of generations (around 200), was unable to resolve conflicts in all but the simplest cases.

iii) Yet another attempt at using GA for fixed channel assignment is described in [29]. The authors describe a modified genetic-fix algorithm that utilizes an encoding technique called the minimum-separation encoding scheme. The number of

1's in each row of the binary assignment matrix A corresponds to the number of channels allocated to the corresponding cell. To satisfy the demand requirement this would normally be constant. Each chromosome is a binary string that represents the matrix A through a concatenation of its rows. The genetic-fix algorithm defines mutation and crossover operators for a binary chromosome in such a way that the number of ones (the "weight" of the vector) is always preserved. The binary encoding is further compressed by taking into account the co-cell constraint that no two channels in the x^{th} cell can be closer than $d_{min} = D_{x,x}$. This would imply that each 1 in each row of the A matrix must be followed by $(d_{min} - 1)$ zeros. The minimum-separation encoding scheme takes advantage of this fact by eliminating $(d_{min} - 1)$ zeros following the 1. Thus the bit string <1000100100> is represented by <1011> when $d_{min} = 3$. This compression reduces the search space further. The implicit encoding of the co-cell constraints and demand requirements in this manner means that the cost function to be minimized only contains the co-channel constraint violation cost.

The genetic fix algorithm was tested on a suite of 5 problems with varying numbers of cells (4, 21 and 25), and numbers of channels (ranging from 11 to 309), along with different compatibility matrices and demand vectors. The results of this algorithm were compared with those from a paper [30] that applied a neural network to the first four of these problems using the frequency of convergence (the ratio of the number of successful results to the total number of runs) as a criterion. The genetic-fix algorithm was found to outperform the neural network in all cases. The fifth problem was for a 21-cell system where the algorithm was able to find a conflict free assignment with fewer channels than previously reported in the literature.

iv) Simulated Annealing is applied to the channel allocation problem in [31]. In the cost function used by the authors, only co-channel interference is considered, and the traffic demand term is explicitly specified. The initial temperature T_o for the annealing was chosen by starting with $T = 0$, and increasing it until the ratio χ of accepted to proposed transitions was between 0.7 and 0.9. The cooling schedule is chosen such that

$$T' = T \cdot \exp\,(-\frac{\lambda T}{\sigma}) \tag{3.8}$$

where is a parameter with value between 0 and 1, and is the standard deviation of the cost at level T. The transitions used to define the neighborhood of an acceptable solution include the basic "flip-flop" (add one channel to a cell, and remove one - preserving the number of channels assigned), and the dense-packing move (find all nearest co-channel cells, pick the channel that is most used in these cells and switch it on at the current cell; randomly select and turn off one of the other channels currently in use in this cell). To further enhance the ability of the annealing algorithm to evade local minima, the authors allow multiple flip-flops in one move (this permits greater movement in the solution space). The paper further discusses how additional constraints imposed on this basic allocation scheme, such as soft interference constraints, hard and soft pre-allocated channels, and minimized spectrum demand, may be incorporated into the cost function.

The Simulated Annealing algorithm is tested on a 7-cell cluster scheme for an in-homogeneous network with 239 cells, 38 channels, some pre-allocated channels and varying demand. Simple (only flip-flop moves) and sophisticated (flip-flop, dense-packing and multiple flip-flops) neighborhood schemes are compared in terms of convergence percentage, final cost, and computing time. As expected, the sophisticated schemes perform significantly better, demonstrating the importance of having a good neighborhood definition.

v) Simulated Annealing and Tabu Search are compared for the CAP in [32]. The cost function to be minimized is given as

$$f'' = \sum_{j \in C} max_{x \in N}\{A_{x,j}\} + \frac{1}{\Psi} f \tag{3.9}$$

where f is as defined in Equation (3.7) with the traffic demand defined implicitly. The first term in the expression attempts to minimize the bandwidth allocated. Ψ is a small positive number that gives a larger weight to the second term that represents the interference constraints.

For the SA, the initial temperature is chosen as in [31] by using the acceptance ratio χ. The cooling schedule is taken from [12]:

$$T' = \frac{T}{1 + \frac{T \ln(1+\delta)}{3\sigma}} \tag{3.10}$$

where δ is a distance parameter chosen to be 0.5 in this paper. For the TS algorithm described in the paper, two tabu lists are maintained - one for the cell (of size 7), one for the channels (size 1). An aspiration criterion is used to allow even tabu moves if they are found to yield lower cost than the current best cost. The dense packing and basic flip-flop moves (described before) are used with equal weights to determine the neighborhood for both SA and TS algorithms.

Applying the algorithms to examples of various sizes (9 to 49 cells with traffic demand between 1 and 10), the authors observe that Simulated Annealing obtains fairly good solutions, while the Tabu Search is extremely ineffective. It took longer than 24 hours on a SUN SPARC 10 workstation for problem instances with 16 cells or more, and on the problem instance of 9 cells it did not match the performance of the SA algorithm in terms of the minimum number of channels used. It is, however, entirely possible that the dismal performance of the TS in this experiment may have been due to an inefficient implementation of the algorithm.

vi) The speculation regarding the poor results of TS in [32] is given some credence by the results in [33] where the use of Tabu Search for frequency assignment is investigated more thoroughly. The formulation of the constraints and the expression for the cost function used in this paper are identical to those given in Equation (3.4). To minimize the number of frequencies (NF) used in a channel assignment, the Tabu Search is carried out at an initially high value of NF. If a conflict-free assignment is found then the value of NF is decreased by 1, and this is repeated until one can no longer find such a conflict-free assignment.

The co-cell constraints are incorporated into a candidate assignment x given as follows:

$$x = \{c_{1,1} \dots c_{1,T_1} \dots c_{x,1} \dots c_{x,T_x} \dots c_{N,1} \dots c_{N,T_N}\} \tag{3.11}$$

such that $\forall x \in 1 \dots N, \forall i, j \in 1 \dots T_x, |c_{x,i} - c_{x,j}| \geq D_{x,x}$, where $c_{x,j}$ is the j^{th} channel allocated to the x^{th} cell. This assignment satisfies the adjacent channel interference constraint by definition.

A neighbor x' of a solution point x can be obtained by changing the value of any channel that has been allocated in x such a way that the new value always satisfies the co-cell constraint. Further a candidate list of new neighbors V^* is specified as follows:

$$V^* = \{x' \, in \, N(x) | x' \, and \, x \, differ \, at \, the \, value \, of \, conflicting \, frequency\} \quad (3.12)$$

The size of this candidate list varies during the search, and helps the search to concentrate on influential moves while reducing the size of the neighborhood. Since only a part of the frequency assignment is changed during each move, the speed of evaluation is reduced by differentially updating the cost as each move is made. The size of the tabu list, k, is made to be proportional to the length of the candidate list at each iteration, $k = \alpha|V^*|$ and thus varies as the algorithm proceeds. Again, the simple aspiration criterion is chosen where the tabu status of a move is cancelled if it leads to a solution better than the best solution encountered so far.

The TS algorithm thus designed was applied on a suite of real-sized problems taken from data provided by the French National Research Center for Telecommunications and compared with a local search algorithm (steepest descent) and the best known results obtained with other methods including Simulated Annealing. For each TS run of 100,000 iterations, the steepest descent algorithm was run 20 times for 5,000 iterations. The local search algorithm repeatedly yielded sub-optimal results while the TS was able to find near-optimal solutions quite speedily for most of the instances selected. In comparing their Tabu Search algorithm with the Simulated Annealing results from [34], the authors showed that the TS algorithm was capable of matching, even outperforming SA in locating the minimal number of frequencies for channel allocation. The TS algorithm was also observed to be faster than the SA. Based upon further experimental evidence, the authors credit the performance of the algorithm to the usage of the conflict-based candidate list strategy and the incorporation of co-cell constraints into the candidate solution points (neither of which was done in [32], where TS did not fare well).

3.3 Optimal Base Station Location

One of the most challenging design problems in the planning and deployment of a mobile radio communication network is deciding on the optimal locations for base stations. Currently the location of base stations is often done in an *ad-hoc* manner via manual inspection of maps showing propagation properties of the area to be serviced. Given a list of potential sites in a service area where base stations may be located, the goal is to use knowledge of the radio propagation characteristics of the area to select sites in such a way as to minimize their number while maximizing coverage in the area. This would reduce network cost and complexity and be a great improvement upon the current methodology for setting up micro-cellular networks. The problem is very similar to the minimum dominating set (MDS) problem from graph theory, which is known to be NP-hard [4].

To perform such an optimization, the cost function necessarily involves the radio propagation characteristics of the area. This may be determined using sophisticated ray-tracing software (which could be a very time-consuming process), or by using empirical propagation models for path loss. Another issue in the design of the cost function is the trade-off between coverage and number of base stations. The more base stations there are, the greater coverage there is, but there is also correspondingly greater radio interference and network cost.

i) Genetic Algorithms are applied to this task in [35]. It is assumed that a list of N possible locations is known beforehand, such that if there were base stations at all those locations, 100% radio coverage is guaranteed. A binary encoding of the problem is chosen as follows: each chromosome consists of N bits, with a one at each bit if there is a base station at the location corresponding to that bit, and a zero otherwise. The simplest GA is first evaluated for the fitness function chosen as follows:

$$f = \frac{R^{\alpha}}{n} \tag{3.13}$$

where R is the radio coverage (the percentage of locations covered by the base stations selected), and n is the number of base stations that are present. α is a parameter that is tuned to favor coverage with respect to the number of transmitters and is assigned a value of 2 in this paper.

Experimental tests of this algorithm were performed on a 70 km x 70 km region in France for $N = 150$, with the radio coverage from each possible location being computed by a radio wave propagation simulation tool. A solution with 52 base stations covering 80.04% was found, but since the algorithm was found to be very slow it was parallelized using the parallel islands model of GA. In this version of the GA, the population is distributed into sub-populations (islands) where they evolve quasi-independently towards the solution, and some individuals in each generation are migrate from one island to another. A nearly linear speedup was observed using this technique and the resultant computational power enabled them to discover a good solution with only 41 base stations that covered 79.13% of the initial covered surface.

ii) The optimal base station location problem for micro-cells in an urban city environment is solved using Simulated Annealing in [37]. An empirically determined model for path loss in urban areas given in [38], which takes as parameters the heights above ground of the mobile and base station antennas, was used to calculate coverage. While the authors concede that the assumptions made regarding path loss are not necessarily valid in reality, they are sufficient to demonstrate the applicability of SA to this problem.

The service area is modeled as a Manhattan-like grid of streets dividing the area into square city blocks (10m x 10m). A power goal is set such that at each grid point i a certain minimum level of radio reception P_i^* is guaranteed. The cost function is then defined as follows:

$$f = \sum_{i \in A} (P_i^* - P_i)^2 \tag{3.14}$$

where A is the set of all grid-points where the actual received power P_i is less than the minimum requirement P_i^*.

Each possible solution is a location x. The neighborhood is defined by allowing the base-stations to make random moves while satisfying any constraints on the location such that the current temperature T at each stage defines the magnitude of these moves. The initial temperature T_o is chosen to be 80m, the cooling schedule is such that $T' = 0.8T$, and at each temperature 100 moves are allowed. The algorithm is stopped when the difference between the maximum and minimum costs at a given temperature is equal to the maximum change in an accepted move at the same temperature. Since the units of cost and temperature are different, the metropolis criterion is modified as follows:

$$P_{x \to x'} = min\left(1, \exp\left[-\frac{(\frac{f(x')-f(x)}{f(x)})}{(\frac{T}{T_o})^2}\right]\right) \tag{3.15}$$

This SA technique was tested on two problems - one involving the location of a single transmitter in a 4 x 4 grid. In this simple problem instance the optimum location could be determined by an exhaustive analysis of all possible grid locations. In all runs (with different initial starting positions) the optimization arrived at a final position within a few meters of the optimum location. Another problem with an 8 x 8 grid and base stations was performed and a figure showing the solution and resultant coverage is presented in the paper. The authors observe that different runs yield different base-station locations with the approximately the same overall costs indicating that near-optimum solutions are not unique.

3.4 Mobility Management

As mentioned before, one of the goals of PCS is to ensure that users are able to communicate with each other wherever they may be. Users should be able to roam at will, crossing cell boundaries system-wide while they are mobile without losing an ongoing call or being prevented from initiating or receiving calls. This necessitates mobility management, since the system needs to track the location of the user in order to provide service. For mobility management purposes, the system is divided into registration areas (RA) consisting of several cells, as seen in Figure 3.3. The user's location is maintained in a location database, such as the Visitor Location Register in GSM mentioned before, and this database is updated each time users enter or leave the RA. This update is referred to as a location

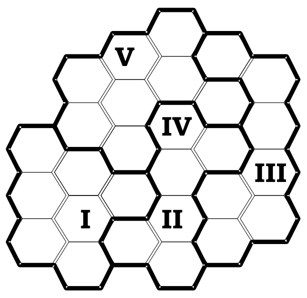


Figure 3.3: Registration Areas in a Cellular System

update or LU. When a mobile station is called, its current RA is determined from such a database, and all base stations in the RA broadcast paging messages over a reserved forward control channel (FOCC) to inform the mobile unit of this call request.

The LUs result in increased network traffic and also place a load on the distributed location databases, resulting in greater complexity of the databases' implementation. The paging messages also consume radio resources and their traffic is often limited by the bandwidth of the FOCC (a condition referred to as paging bound). It is thus desirable to design the system in such a way as to reduce both LU and paging costs. There is a tradeoff between the LU and paging costs which varies with the size of the RA: If the RA is large, there are fewer inter-RA crossings resulting in a lower LU cost, but the number of base stations that need to be paged increases correspondingly. Genetic Algorithms have been used to design optimal RA to reduce LU cost while maintaining paging bound constraints in [39]. Within each RA, only one mobile station responds to the multiple pages transmitted by base stations in the RA. To reduce the paging cost, it is possible to subdivide the RA into paging zones ordered based on a pre-determined probability of locating the mobile user at different locations within the RA. The optimal planning of such paging zones using GA is treated in [40].

i) The RA planning in [39] is based on the assumption that the cell planning has already been done and the LU and paging traffic have been estimated for each cell. It is assumed that the paging bound B for each RA is fixed. The RAs are planned so that they each contain a disjoint set of cells. The task is to design the RAs to minimize the following cost function:

$$f = \frac{\alpha_1}{2}\sum_{i=1}^{M} LU_i + \alpha_2 \sum_{i=1}^{M} max(0, (P_i - B)) \tag{3.16}$$

where P_i and LU_i are the paging traffic and LU traffic respectively in the i^{th} registration area, and M is the total number of registration areas in the system. The second term in Equation (3.16) is non-zero if the paging bounding is exceeded in the i^{th} registration area due to an increased number of cells in it . and are constants used to weigh the relative importance of the LU cost and the paging cost.

A binary representation chosen for the chromosomes is based on the cell borders. The borders are numbered sequentially and the corresponding bit is one if that particular cell border is to be a border between two adjoining RAs, and zero otherwise. This representation facilitates the evaluation of the location update cost. The first term in Equation (3.16) can be rewritten for this border notation as follows:

$$f = \frac{\alpha_1}{2}\sum_{j=1}^{n} v_j w_j + \alpha_2 \sum_{i=1}^{M} max(0, (P_i - B)) \tag{3.17}$$

where n is the total number of borders, w_j the crossing intensity of the j^{th} border and v_j the value of the j^{th} bit of the chromosome being evaluated (i.e. a determination of whether the j^{th} border is an inter-RA border). It is also shown in the paper how additional terms may be incorporated into the cost function to deal with useless location updating (when no calls arrive between successive LUs) and presets that arbitrarily force certain borders in the chromosome to be 0 or 1. The standard GA operators of mutation and crossover are used. The mutation rate (initially 0.02) is halved after several generations to increase convergence. The GA is terminated after 1000 generations.

The GA is compared with hill-climbing for five systems with varying size (19 to 91 cells) where the paging cost for each cell and crossing intensity for each border were generated with a normal distribution. The result of RA planning over 100 runs shows that GA's consistently outperform hill-climbing providing improvements ranging from 10-30%. The authors conclude that GAs appear to be a valuable approach to the planning of registration areas in PCS networks.

ii) A tracking strategy utilizing the mobility patterns of individual mobile stations is used in [40] to plan the partition of RAs into paging zones to minimize paging cost. First a multilayered model is developed based on mobile-phone usage for different times of activity during a typical workday for each user - work, home, social. Each mobile user's activity is monitored for each such layer to develop statistical mobility patterns that describe the likelihood of locating each user in a particular cell during a particular time of the day. Then the cells in the RA are partitioned into paging zones for each user k, and layer j such that each zone consists of cells with similar likelihood of locating that user:

$$P_{k,j} = \{P_{k,j,l}\} \tag{3.18}$$

These zones are arranged in order of descending likelihood of locating the user

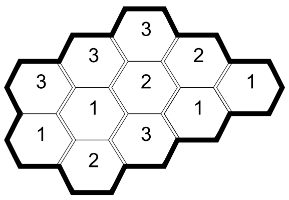


Figure 3.4: Cells belonging to three Paging Zones within a Registration Area (For a Unique User)

denoted by $prob(Pk, j, l)$ - the probability of locating the k^{th} user in activity layer j in the l^{th} zone. Now when a call is received for the user in the RA, the zones are paged in this order - which is more efficient than paging the entire area. The cost function derived in the paper for each paging zone assignment is based on the mobile location probability for each zone in a partition:

$$f = N(P_{k,j,1}) + \sum_{l}(1 - \sum_{i=1}^{l-1} prob(P_{k,j,i})) \cdot N(P_{k,j,l}) \cdot \alpha \cdot \beta \tag{3.19}$$

where $N(P_{k,j,l})$ represents the number of cells in the l^{th} zone, α the traffic in the FOCC per page, and β the traffic in the MSC per page. Since paging zones are unique for each user, this optimization must be carried out for each user separately.

The encoding chosen for the GA utilizes an integer representation. The cells in the RA are numbered sequentially and correspond to a locus on the chromosome. The value of the chromosome at that locus is the paging zone to which the cell belongs. Thus the string "11231" indicates that the first, second and fifth cells are in zone 1, the third cell in zone 2 and the fourth cell in zone 3. Zone 1 has the highest mobile location probability. The cells for a given paging zone need not be adjacent as seen in Figure 3.4. The standard genetic operators of crossover and mutation are used. The algorithm is tested for different numbers of paging zones (from 1 to 4) and it is observed that the greater the number of zones, the lower the paging cost. Even a 2-zone partition is observed to have a degree of magnitude lower cost than the traditional area-wide paging scheme.

3.5 Call Management

Another issue of mobile communication system planning is call management. One aspect is this is the determination of a call-admission policy - deciding under what conditions a new call to a mobile in a particular cell should be accepted. Allocating channels to every user whenever they are available (the admit-if-possible or AIP strategy) is not necessarily an optimal strategy because this may result in an inability to accept hand-off calls from a neighboring cell.

i) The problem of evolving an admission call policy for each cell based only on local state information (restricted to the cell and its neighbors) is considered in [41] using Genetic Algorithms. In a mobile communication system it is generally considered better to block a new call than drop an ongoing one. The cellular system performance (which depends upon the call admission policy) is the cost function to be minimized:

$$f = P_b + \omega P_h \tag{3.20}$$

where P_b is the new call blocking probability and P_h the handoff dropping probability. The factor ω is used to indicate the extent to which dropped calls are considered less desirable than blocked calls.

In each cell, the number of states refers to how many of its C channels are occupied. For a linear cellular system with N cells, the total number of global states is

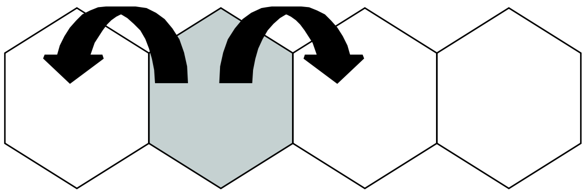


Figure 3.5: Linear Cellular System N=4, k=1 (2 Neighboring Cells Considered)

$(C+1)^N$ for each decision. The policy is constructed to consist of three binary decisions: 1) admit a new call 2) admit a handoff call from the left cell and 3) admit handoff from right cell. In a global admission policy there is one decision for each global state (thus $3(C+1)^N$ decisions in all). The states in a global admission policy satisfy the Markov property since each succeeding state depends only upon the previous state of the network. Markov decision process (MDP) based techniques such as Dynamic Programming [51] may be used to solve such a problem but the state space becomes too large for moderate sized networks. If a local policy is considered, and only state information from $2k$ nearest neighbors are used, the state space is $3(C+1)^{2k+1}$. In this case, however, the states are no longer Markov states and MDP techniques are no longer applicable. The state space is still very large: for an example with 9 channels, a policy with $k = 1$ (using information from the current cell and both immediate neighbors in a 1-D linear cellular system) requires 3000 bits (1 x 10903 possibilities).

A standard GA is used in [41], where each chromosome represents a local policy, with admit or reject decisions for the new call and handoff requests for each local state of the system. The local policies are evaluated via Monte Carlo simulation using the cost function described in Equation (3.20). The algorithm was tested on a small 4-cell system for both $k = 0$ and $k = 1$ (seen in Figure 3.5) to compare it with the optimal solution for a global policy obtained with MDP and the AIP strategy. As the factor w increases, performance is observed to become worse since more global information is needed to better determine handoff possibilities. But in all cases the solutions provided with GA give much better system performance than the AIP strategy. For a larger system with 16 cells and 9 channels the GA (for cases $k = 0$, and $k = 1$) was compared with AIP strategy and the best single-threshold handoff reservation policy (having a threshold t such that if cell occupancy $s > t$, new calls will not be accepted). Again the GA-based solutions (even for $k = 0$) were much better than either of these. Upon examination of the GA-based solutions for $k = 0$, the authors observed that they tended to suggest two-threshold policies with thresholds t_1 and t_2 such that new calls are accepted if the occupancy s is less than t_1 or greater than t_2, but not if it is between the two. The authors suggest an explanation for the result: when the cell is nearly empty ($s < t_1$) new calls may be admitted without fear of not having room for handoff calls. When it is a little full ($t_1 \leq s \leq t_2$) handoffs are expected and so some channels are reserved for handoff. When the cell is nearly completely occupied ($s > t_2$), it becomes less likely that the neighboring cells are occupied and hence there is a reduced handoff possibility and new calls are accepted again.

The GA was thus able to identify a novel policy structure. Based on a comparison of performances, the authors suggest that since the state space is considerably smaller for policies with $k = 0$ than $k = 1$ (30 bits vs. 3000 bits to represent the policy), and they offer nearly similar results, it is advisable to search for single-cell occupancy based policies.

3.6 Optimal Multi-user Detection in CDMA

The applications thus far presented are all geared towards high-level design of mobile communication networks. We now consider the difficult low-level optimization problems in mobile communications that have been tackled using global search techniques. One such problem is that of optimal multi-user detection in Code Division Multiple Access (CDMA) systems. Spread spectrum CDMA is a fairly new wide-band technique used to share the available bandwidth in a wireless communication system. In CDMA the message signal is multiplied by a large-bandwidth *spreading signal* - a pseudo-noise sequence with a chip rate much greater than the rate of the message. Each user is assigned a unique PN codeword that is nearly orthogonal to the other codewords. For the receiver to detect the message signal intended for the user, it must know the corresponding unique codeword used by the transmitter. In the single-user receiver the received waveform is input to a correlating matched filter and the signals of the other users are treated as noise (along with the noise due to the channel). But because the other codewords are still somewhat correlated with that of the receiving user's codeword, this receiver is not optimal in minimizing errors in detection. It has been shown that the optimal receiver would be one that detects the messages for all users simultaneously, i.e. one that performs simultaneous multiuser detection [42]. Formally, considering a synchronous CDMA channel with additive white Gaussian noise (AWGN) characteristics, given that the k^{th} user (of the K users sharing the channel) is assigned a codeword , the signal $r(t)$ at the receiver is:

$$r(t) = \sum_{k=1}^{K} b_k s_k(t) + \sigma n(t) \ , \ t \in [0, T] \tag{3.21}$$

where n(t) is a unit spectral density white Gaussian process and represents the transmitted bits in the message for the kth user. The output of a bank of matched filters that correlate this signal with the codewords for each user is given by:

$$y_k = \int_0^T r(t) \cdot s_k(t) dt \tag{3.22}$$

It has been shown that the output vector $\overrightarrow{y} = [y_1, y_2 \ldots y_K]$ thus obtained may be used to determine the transmitted message vector $\overrightarrow{b} = [b_1, b_2 \ldots b_K]$ in an optimal manner [41]. If H is the nonnegative matrix of cross correlation between the assigned waveforms:

$$H_{i,j} = \int_0^T s_i(t) s_j(t) dt \tag{3.23}$$

then

$$\overrightarrow{y} = H\overrightarrow{b} + \overrightarrow{n} \tag{3.24}$$

(25) where $\overrightarrow{n}$ is a Gaussian K-vector with covariance matrix equal to $\sigma^2 H$. The optimal multiuser detection procedure determining the message that was most likely sent is to determine the vector $\overrightarrow{b}$ that maximizes the following expression:

$$f = 2\overrightarrow{y}^T \overrightarrow{n} - \overrightarrow{b}^T H \overrightarrow{b} \tag{3.25}$$

This combinatorial optimization problem is known to be NP-hard, as the search space of the possible multi-user message increases exponentially with the number of users.

i) [43] describes the application of GA to optimal multiuser detection. It is assumed that the bits are being decoded sequentially (one bit for each user at any time) using the Viterbi algorithm. To speed up the search and reduce the computational complexity the authors suggest reducing the number of bits by utilizing the information given by the output of the matched filter bank. A confidence measure based on the probability of a bit error in single-user detection is used to determine if some of the bits may be considered as detected. The bits b_k are thus divided into two groups - the group of detected bits and the group of unknown bits. The search space is then restricted to finding values only for the unknown bits. In the Genetic Algorithm each chromosome consists of the number of bits that are still unknown. The standard mutation operation, uniform crossover and an elitist selection scheme were used. A total of 20 chromosomes are present in each generation and for each step of the detection process only 6 generations were performed (only 120 evaluations are performed in the optimization to detect the unknown bits).

To examine the proposed detector, Monte Carlo simulations were carried out comparing the GA-based detector with the linear decorrelating detector and the linear MMSE detector for $k = 20$ users. The authors show that the performance of the GA-based detection is very close to that of the MMSE detector and better than the decorrelating detector, and further that the proposed detector is actually computationally faster than these two linear detectors.

ii) A very brief paper [44] also describes the application of Genetic Algorithms to this problem. The authors reported using Monte Carlo simulations to study the performance of the genetic detector on a problem with 20 users and spreading sequences of length 31. The algorithms were compared for different types of initial guesses (random and those generated by sub-optimal detectors). They concluded that the algorithm was not robust when random initial guesses were used, but was able to improve the detection performance when combined with such sub-optimal detectors.

3.7 TDMA Frame Pattern Design

Besides CDMA, another multiple access scheme very commonly used in mobile communication systems is Time Division Multiple Access (TDMA). In TDMA the information for different users is carried in separate time slots in each frame of data. One of the goals of PCS is to be able to carry multimedia multi-rate information on the network. This is easily done in TDMA because users have the flexibility of determining different rates and error-control coding for traffic of different kinds of information- such as systems that carry both voice and data.

One way of integrating voice and data in TDMA is to divide the frame into fixed sub-frames for each. But this *frame pattern* has the significant disadvantage of not being able to adapt to traffic changes. A dynamic scheme that allows a data user to borrow voice slots when needed improves utilization. In schemes where a contention-based transmission technique is used for data, the positions of available data slots within a frame affect the performance of the system.

i) In [45] the problem of designing frame patterns to maximize throughput is considered and solved using a Simulated Annealing algorithm. It is assumed that each frame consists of N independent slots, N_t of which are assigned for data service (including nominal data slots and unassigned voice slots) while the rest are assigned for voice traffic. The frame pattern of the integrated TDMA system is defined as

$$\overrightarrow{x} = [x_1, x_2, ...x_{N_t}] \tag{3.26}$$

where $x_i \geq 1$ represent the number of slots between successive data slots. It is assumed that the slotted ALOHA random access protocol [7] is used for data transmission, that the queuing behavior of data for a given pattern is able to reach steady state and that data traffic is a Poisson process with mean arrival rate G packets/slot. For a given frame pattern, the mean data throughput is the function to be maximized. To perform minimization, the cost function may be taken as the negative of this throughput measure, and is therefore given by:

$$-\frac{G}{N_t}\sum_{i=1}^{N_t} x_i \cdot \exp(-G \cdot x_i) \tag{3.27}$$

The number of frame patterns for a given N_t and N can be very large indeed. For a TDMA frame with $N = 60$, and $Nt = 10$, there are as many as 7.5 x 1010 possibilities. Since a brute force technique is not possible in this huge search space, global search techniques are ideal for this NP-hard problem.

For the SA, the initial temperature T_o is chosen to be $10\sigma_\infty$ (the standard deviation of the cost of explored states when T is infinite). The neighborhood function is defined by a simple move (add 1 to one of the x_i, and subtract it from another randomly) and a complex move which is simply a succession of simple moves. A further adaptive scheme is used to determine the selection probabilities of the moves based on data collected during the execution of the algorithm. The standard metropolis criterion is used. The selected cooling schedule is similar to that in Equation (3.8), with an additional term ensuring a minimum rate of cooling. The algorithm is stopped when it is found that all the accessed states are of comparable costs at the given temperature.

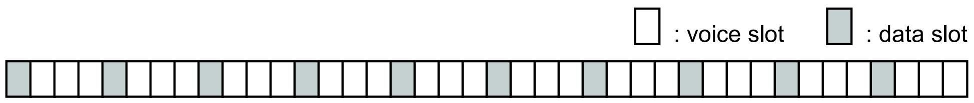


Figure 3.6: Optimal TDMA Frame Pattern ($N = 40$, $N_t = 10$)

The algorithm was tested for various loads G for a frame with $N = 40$ and N_t values of 5, 10 and 15. As anticipated, it was shown that there is an optimal frame pattern for a given G and N_t but none for all G and N_t. Due to the restriction of integer solutions for the optimal frame pattern, it was found that the optimal frame pattern is insensitive to small changes in G and thus there are a finite number of total optimal frame patterns. The maximum throughput of 0.368 occurs at $G = Nt/N = 0.25$ when the frame pattern is uniformly distributed - each slot separated from the succeeding one by an inter-distance of 4 when $N_t = 10$ (shown in Figure 3.6). Comparing the results with random frame patterns, throughput gains ranging from 10% to 120% were observed by the authors.

3.8 Data Equalization

One major problem with digital communications channels is the distortion of data due to inter-symbol interference (ISI) [24]. The limited bandwidths and frequency selective fading (as observed in mobile radio environments) of these channels cause transmitted symbols to be smeared in time and interfere with adjoining bits. The effect of ISI can be reduced by *equalization*. An equalizing filter compensates for the characteristics of the channel and corrects the distortion, reducing errors in communication.

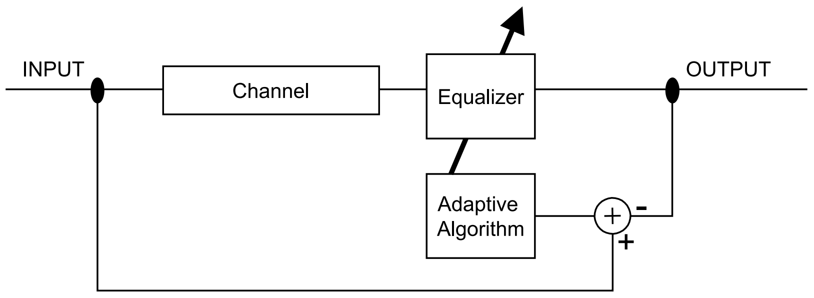


Figure 3.7: Adaptive Data Equalization

Most equalizers utilize filters where the output is non-recursively dependent upon present and previous values of the input and exhibit a finite impulse response (FIR). Recursive infinite impulse response (IIR) filters with feedback have outputs that depend not only on previous values of the input but also those of the output. Though susceptible to instability, IIR filters can perform better when the ISI extends over a long duration and typically require fewer weights (filter coefficients) to be adjusted to perform as well.

i) Figure 3.7 shows the block diagram of an adaptive equalization scheme. The problem of determining the filter coefficients of an adaptive IIR data equalizer is a difficult one. A GA-based algorithm for such equalization is presented in [46]. The chromosomes in the GA represent the filter coefficients (both feedback and feed-forward). The GA used is a simple binary encoded one. The cost function to be minimized is the mean square error (MSE) between the output of the equalizer and the input to the channel, which is known since a training data sequence is used to run the adaptive algorithm. The authors use a 6th order filter and test it on three different channel instances with varying characteristics. In these channels the bit-error rate due to ISI in the data transmitted without equalization ranged from 26% to 32%. After the adaptive equalization was performed, this error dropped to 0% in the first two cases, and 3% in the third case (which had channel noise added in besides the ISI distortion), demonstrating the effectiveness of the genetic adaptive algorithm.

3.9 Blind Channel Estimation

Adaptive channel equalizers utilize a training sequence at the start of transmission. But there are cases when the equalizers may have to start or restart without a training sequence - this is referred to as blind equalization. In the fading/multipath channels characteristic of mobile communications, such a restart may be required when there is temporary path loss due to a severe fading. Under such conditions blind channel identification schemes attempt to determine the coefficients of the channel from the statistical properties of the received signals, specifically higher order cumulants (HOC). This knowledge of the channel may then be used to design the equalizer. The application of GA and SA to the problem of blind channel identification using HOC fitting is presented in [47], [48], [49].

If the message transmitted is $s(k)$, the received signal $r(k)$ can be modeled as the result of ISI and noise as follows [47], [48] :

$$r(k) = \sum_{i=0}^{n_a} a_i s(k-i) + e(k) \tag{3.28}$$

The goal is then to determine the channel coefficients a_i (the channel is assumed to be of order n_a using only $r(k)$ and some knowledge of the statistical properties of $s(k)$. It has been shown in [49] that the problem can be expressed as the minimization of the following function:

$$f = J_4(a) = \sum_{\tau=-n_a}^{n_a} \left(C_{4,r}(\hat{\tau}, \tau, \tau) - \gamma_{4,s} \sum_{i=max(0,-\tau)}^{min(n_a, n_a-\tau)} a_i a_{i+\tau}^3 \right)^2 \tag{3.29}$$

where $C_{4,r}(\hat{\tau}, \tau, \tau)$ is the time estimate of the diagonal slice of the fourth-order cumulant for $r(k)$ and $\gamma_{4,s} = C_{4,s}(\tau_1, \tau_2, \tau_3)|_{\tau_1=\tau_2=\tau_3=0}$ is the *kurtosis* of $s(k)$.

i) In [47], the minimization is approached using a version of Genetic Algorithms called μGA - where the population size is much smaller than usual and the population is reinitialized randomly while retaining the best individual found up to that point after each run converges. The standard crossover operator is used, but no mutation operation is performed since the re-initialization procedure itself introduces diversity in the process. This process is repeated until no further improvement is seen. A 16-bit number between -1 and 1 represents each channel coefficient, and the encoding is such that these coefficients are arranged linearly in the chromosome.

The algorithm was tested on two channels with known coefficients. 50,000 noisy received data samples were used to compute the time estimate of the fourth-order cumulant. The final results were the average of 100 different runs where each run utilized a different sequence of received data samples and different initial populations. It was observed that the solutions always converged quickly to a value close to the global optimal channel estimate. The authors remark that the method is accurate and robust as demonstrated by the small standard deviations of the estimated channel coefficients over the different runs. They compare these results with results from a prior SA-based approach in terms of the number of function evaluations required for optimization. It is noted that the SA-based approach would have taken nearly six times as long for these problems. In a related paper ([49]) on GA-based blind channel estimation, third-order cumulant statistics are utilized instead and the simple Genetic Algorithm is used.

ii) The application of an adaptive Simulated Annealing (ASA) algorithm to blind channel identification with HOC fitting is described briefly in [48]. The function to be minimized is the same as given in Equation (3.29). The ASA algorithm, described in this paper, is a technique to speed up the convergence rate of Simulated Annealing by using estimates of the partial derivatives of the cost function (taken with respect to the solution coefficients) to modify the annealing rate every few moves. The algorithm was tested by applying it to a channel with known coefficients. As expected, ASA was observed to converge to the optimal solution much faster than the standard SA.

3.10 Summary

This chapter has surveyed the application of three global search techniques - Genetic Algorithms, Simulated Annealing, and Tabu Search – to difficult optimization problems in the subject of mobile communications. The papers described are all very recent, and they cover a broad spectrum of problems in mobile communications, from network planning and design issues to lower-level problems like detection of transmitted data and estimation of channel properties.

In the surveys, some details of the implementations of the various techniques have been provided - this becomes a key issue with these algorithms as they are very general in their simplest form and there are a large number of parameters that may be selected to create different implementations. It must be observed, however, that these techniques appear to be quite robust and offer good results for a wide range of implementations as seen for example in the papers on channel assignment. There is, moreover, no empirical evidence of the inherent superiority of one of these techniques over the others for any given global optimization problem. In the few cases where the results of one technique are compared with another, the question of whether each technique was implemented to ensure the fastest convergence possible arises and remains unresolved because of the very nature of these algorithms. Researchers would do well not to ignore the possibilities inherent in each of these techniques and make their decisions on which technique to use based on at least a cursory comparison of their efficiency and ease of implementation.

The speed of convergence becomes important when these techniques are applied to time-critical problems where fast, online solutions are required. So far they have not been applied to such problems in wireless telecommunications because of the large number of function evaluations that are often required to find optimal or near-optimal solutions using these techniques. One of the trends of future research in this area would be the extension of the use of these algorithms to such optimization problems as well. In time, more problems in mobile communications may be identified where such techniques outperform existing schemes. It is already certain that the early success of these approaches will lead to their integration into the design process for future PCS networks.

Chapter 4

Analysis of Search Techniques

To analyze the performance of search algorithms on optimization problems, there are essentially three approaches - worst-case complexity analysis, complete theoretical analysis, and average-case experimental analysis. Some description of complexity analysis for search techniques are provided in [17]. Worst case results are somewhat unsatisfactory because for many real world problems of interest search algorithms have better than the predicted worst case. For NP-complete problems, for example, in the worst case these algorithms would run in exponential time, yet empirical evidence shows that in practice they obtain near-optimal solutions in polynomial time.

4.1 A Markov-based Theoretical Analysis

It is extremely difficult to theoretically predict the performance of search algorithms on real-world problems because this performance depends on (i) a generally unknown and exponentially large search space for the problem, (ii) the neighborhood definition used, and (iii) the various parameters of each algorithm.

It is, however, possible to perform some theoretical analysis for simple cases of search problems. A probabilistic framework for such a complete analysis using Markov chains is described here briefly [1]. The search algorithms are all iterative techniques. Let x_i be the point in the search space visited at the i^{th} iteration, with a cost function value of $f(x_i)$. Further let x_i^* denote the point with the lowest cost function seen in iterations 0 to $i-1$. The cumulative distribution function (cdf) for the cost function value of the lowest point seen to date at the n^{th} iteration, $f(x_n^*)$ is given by:

$$\begin{aligned} P\{f(x_n^*) \leq a\} \quad &= P\{\exists f(x_i) \leq a, \forall i = 0 \; to \; n\} && (4.1) \\ &= 1 - P\{f(x_i) > a, \forall i = 0 \; to \; n\} && (4.2) \\ &= 1 - P\{f(x_0) > a\} P\{f(x_1) > a | f(x_0) > a\} \ldots \\ &P\{f(x_n) > a | f(x_{n-1}) > a, \ldots f(x_0) > a\} && (4.3) \end{aligned}$$

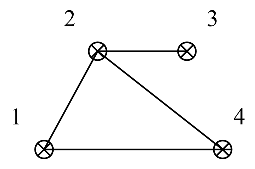
We may further take into account the fact that the Markov property holds, i.e. the current point in the search depends only upon the previous point, not the history of the search. This is true for all memory-less neighborhood based search techniques like Random Walk, Simulated Annealing and Genetic Algorithms (Tabu Search can only be incorporated into this framework if the Tabu List is included in the description of the Markov states). The cdf of the best cost function evaluation to date can then be calculated by utilizing the markov one step state transition probabilities at each step. The cdf in equation 4.3 represents a full probabilistic

[1]this was developed jointly by Xi Xie and Bhaskar Krishnamachari at Cornell University and is still under development

description of the performance of a search algorithm and can be used to determine for example the expectation and variance of the solution found after n steps for a given algorithm. But upon examining this equation, it becomes apparent why a complete theoretical analysis is really difficult. To solve for the cdf, we need to determine the conditional probabilities $P\{f(x_i) > a | f(x_{i-1}) > a \ldots f(x_0) > a\}$ for all possible cost values a. Even if we assume that the search can be treated as a homogeneous markov chain, i.e. the state transition probabilities do not change with the iteration (a limiting assumption which is not true for Simulated Annealing for example where the cooling schedule lowers the probabilities of uphill moves as the search proceeds), we need to know $\mathbf{P}$, the one-step state transition probability matrix. $\mathbf{P}$ is of size $\mathcal{Y}$x$\mathcal{Y}$ where $\mathcal{Y}$ is the number of distinct cost function values in the exponentially large search Space. Determining $\mathbf{P}$ is quite hard for real world problems, because it depends not only on the problem space, but also the neighborhood definition and the algorithm being analyzed. Further, the complexity of solving for the cdf also increases exponentially as the number of iterations for which the performance analysis is desired. It turns out that deriving the cdf becomes impossible for any but the simplest cases involving search spaces with no more than about 4 to 6 points.

We present briefly the results of such an analysis for three iterations of Random Walk algorithms, for every possible Neighborhood definition that can be constructed for a search space of size 4 (subject to assumptions described below). Even this simple case offers some worthwhile insights.

We assume that the search space consists of four points $x1 \ldots x4$ with different cost values. If we restrict ourselves to the Random Walk algorithms which do



$$\mathbf{N} = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{pmatrix} \qquad \mathbf{P} = \begin{pmatrix} (1-p) & 0.50p & 0 & 0.50p \\ 0.33 & 0.67(1-p) & 0.33p & 0.33p \\ 0 & 1 & 0 & 0 \\ 0.50 & 0.50 & 0 & 0 \end{pmatrix}$$

Figure 4.1: A Neighborhood Definition for Search Space of Size 4 and the corresponding State Transition Probability Matrix

not depend upon the absolute value of the costs, we may assume without loss of generality that their cost functions are given as follows:

$$f(xi) = i \tag{4.4}$$

Thus, in this search space, $f(x1) = 1$ is the global minimum. The neighborhood definition in general can be described by an adjacency matrix $\mathbf{N}$ or directed graphs where an arrow from point xi to xj indicates that $xj \in N(xi)$. If we assume that the neighborhood definitions are symmetric, i.e. $xj \in N(xi) \iff xi \in N(xj)$, then $\mathbf{N}$ is symmetric and the neighborhood definition can be described by an undirected graph. It is further assumed that $xi \notin N(xi)$ and that the neighborhood graph is fully connected (i.e. it is possible to get from any point $xi \in S$ to any another point $xj \in S$ in a finite number of steps). Under these assumptions there are 38 possible neighborhood functions for the given search space. Figure(4.1) shows the graph and adjacency matrix $\mathbf{N}$ for one of these neighborhood definitions. The different Random Walk algorithms differ in the probability p with which uphill moves are accepted. For a given $\mathbf{N}$, the one-step Markov state transition probability matrix $\mathbf{P}$ can be determined as a function of p as shown in Figure(4.1). For each neighborhood definition, $\mathbf{P}$ may then be used to evaluate the cdf given in equation (4.3), from which the expected value of the best cost to date may be calculated for each iteration. These calculations were performed for all the 38 neighborhoods for p ranging from 0 to 1 under the assumption that the initial point is picked randomly.

Figures (4.2) - (4.6) present the results. Each subfigure corresponds to a different neighborhood definition with the given adjacency graph. The x-axes represent p, the uphill probability and the y-axes represented the expected value of the best cost seen to date for the first three iterations. As expected, even for a given problem it can be seen that the performance of the algorithms depends upon the definition of the neighborhood. It is not always the case that greedy search ($p = 0$) outperforms the purely random search ($p = 1$). Another interesting result is that for the last neighborhood considered in figure (4.6) where every point is adjacent to every other point ($N(xi) = S, \forall i$), all Random Walk Algorithms perform equally well. This can be understood as follows - given a neighborhood definition, the algorithms differ only in the selection criterion, i.e. how the next point is to be selected. When every point in the space is a neighbor, the choice of the current point does not influence the next point to be considered since all points are considered with equal probability. Thus the best point seen up to any iteration is independent of the selection criterion, and hence all algorithms perform equally well, or just as poorly as the purely random search where at each step we try to "guess" at the answer.

This is actually a very broad result, and applies to all search spaces and algorithms when the neighborhood definition is such that at each step a completely random point is considered, i.e. $N(x) = S, \forall i, x \in S$. We have thus an example of a neighborhood definition under which all algorithms perform equally well.

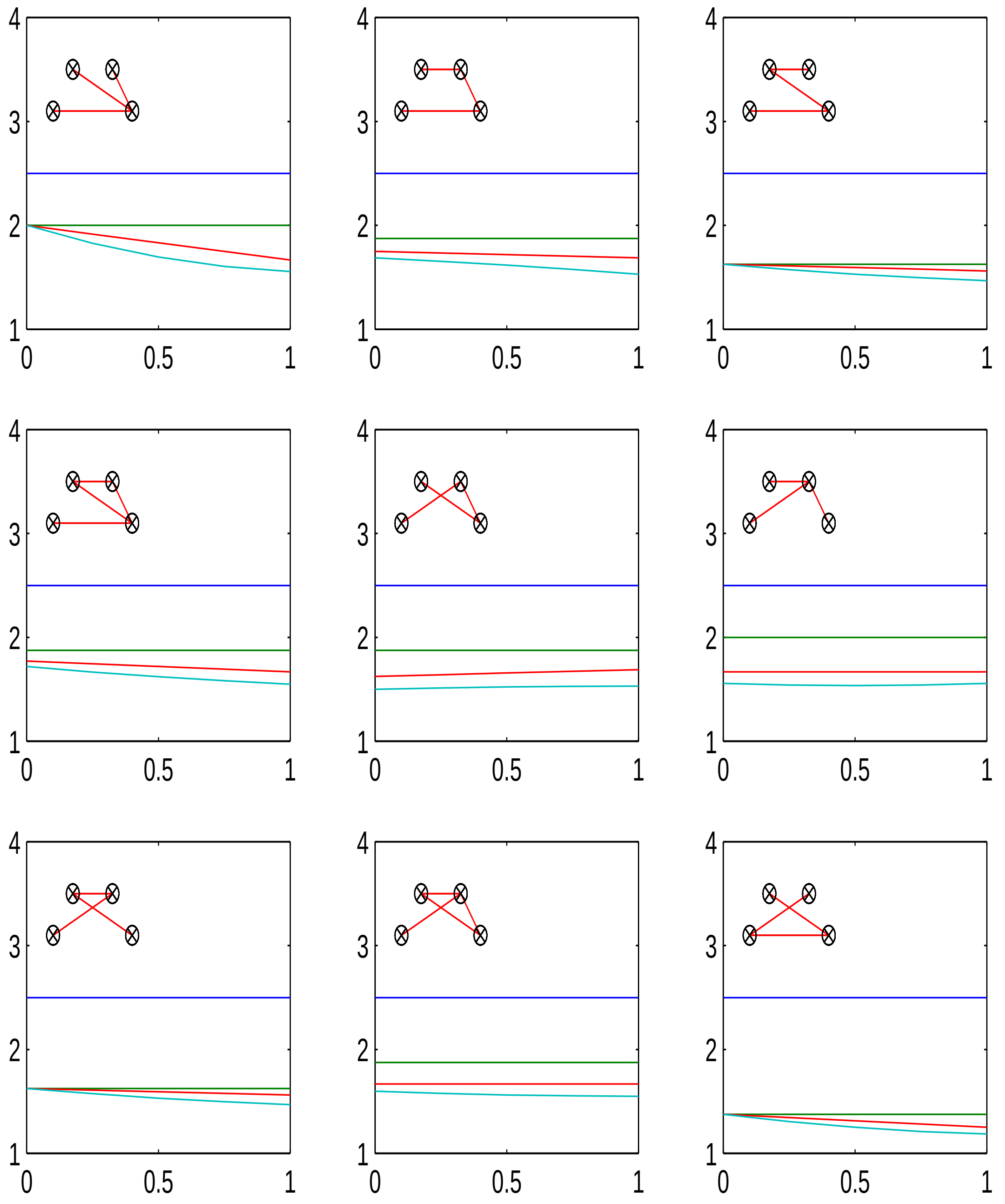


Figure 4.2: Expected Best Cost after 0,1,2,3 Iterations for Random Walk Algorithms for different Neighborhood Definitions (1)

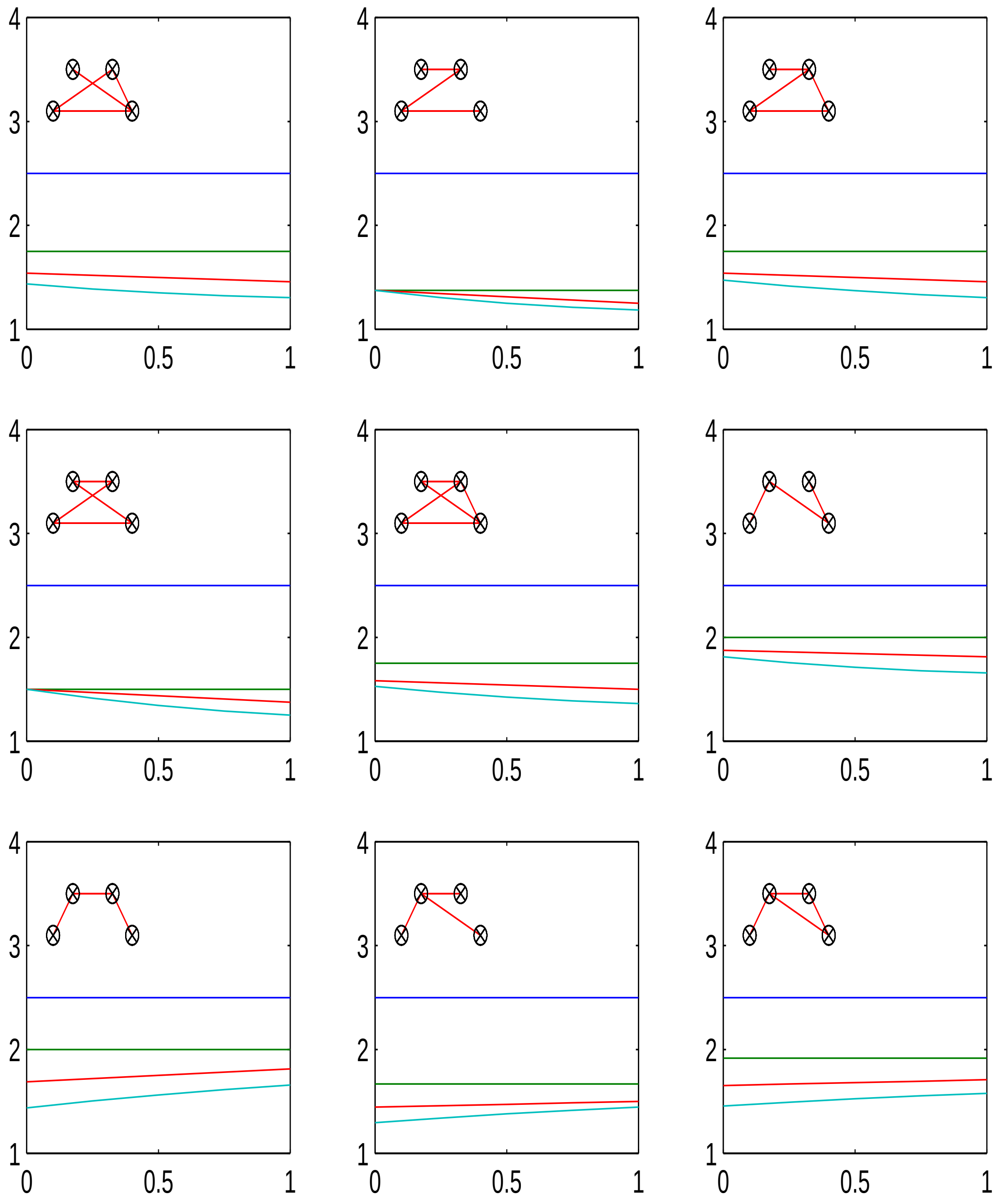


Figure 4.3: Expected Best Cost after 0,1,2,3 Iterations for Random Walk Algorithms for different Neighborhood Definitions (2)

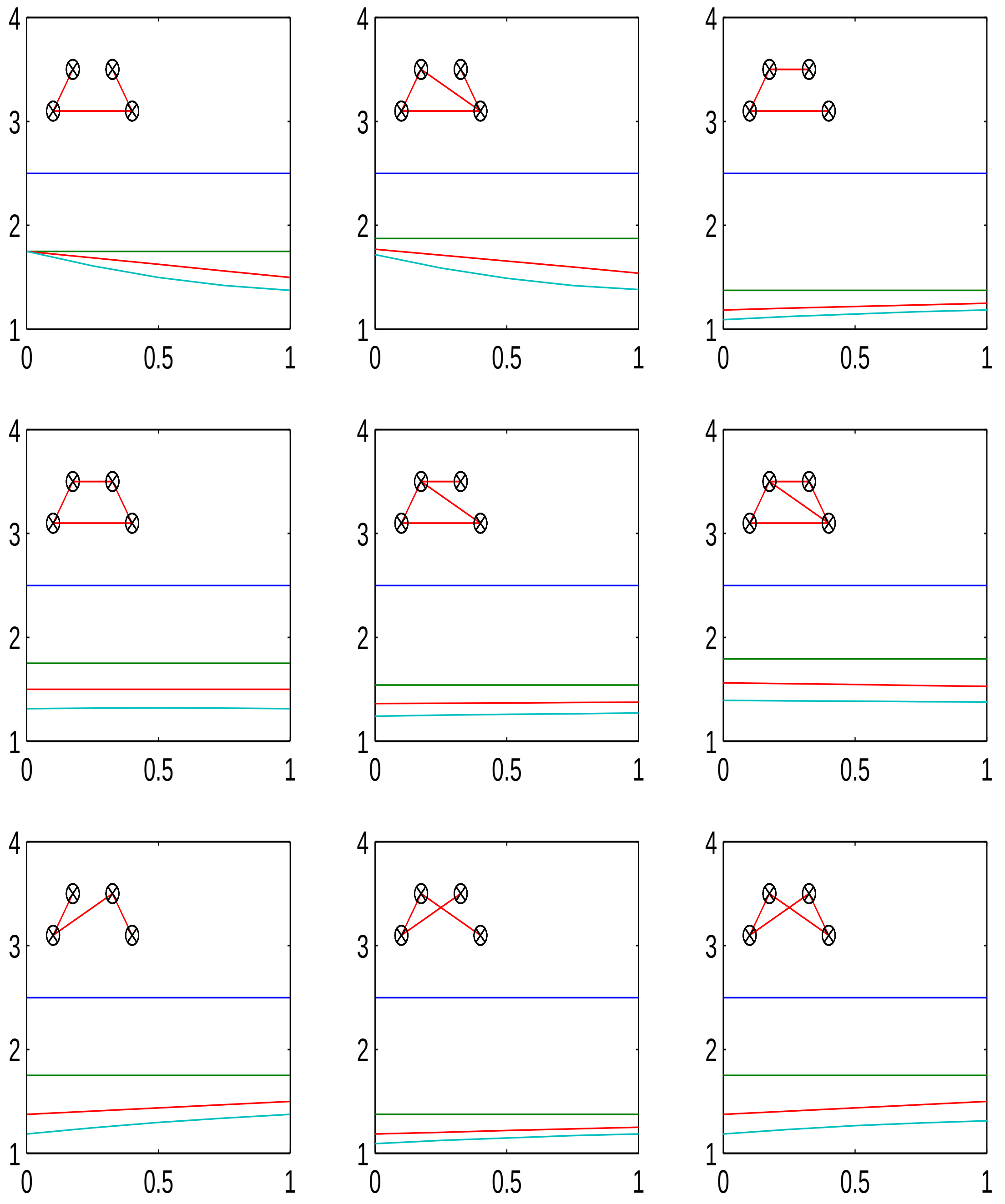


Figure 4.4: Expected Best Cost after 0,1,2,3 Iterations for Random Walk Algorithms for different Neighborhood Definitions (3)

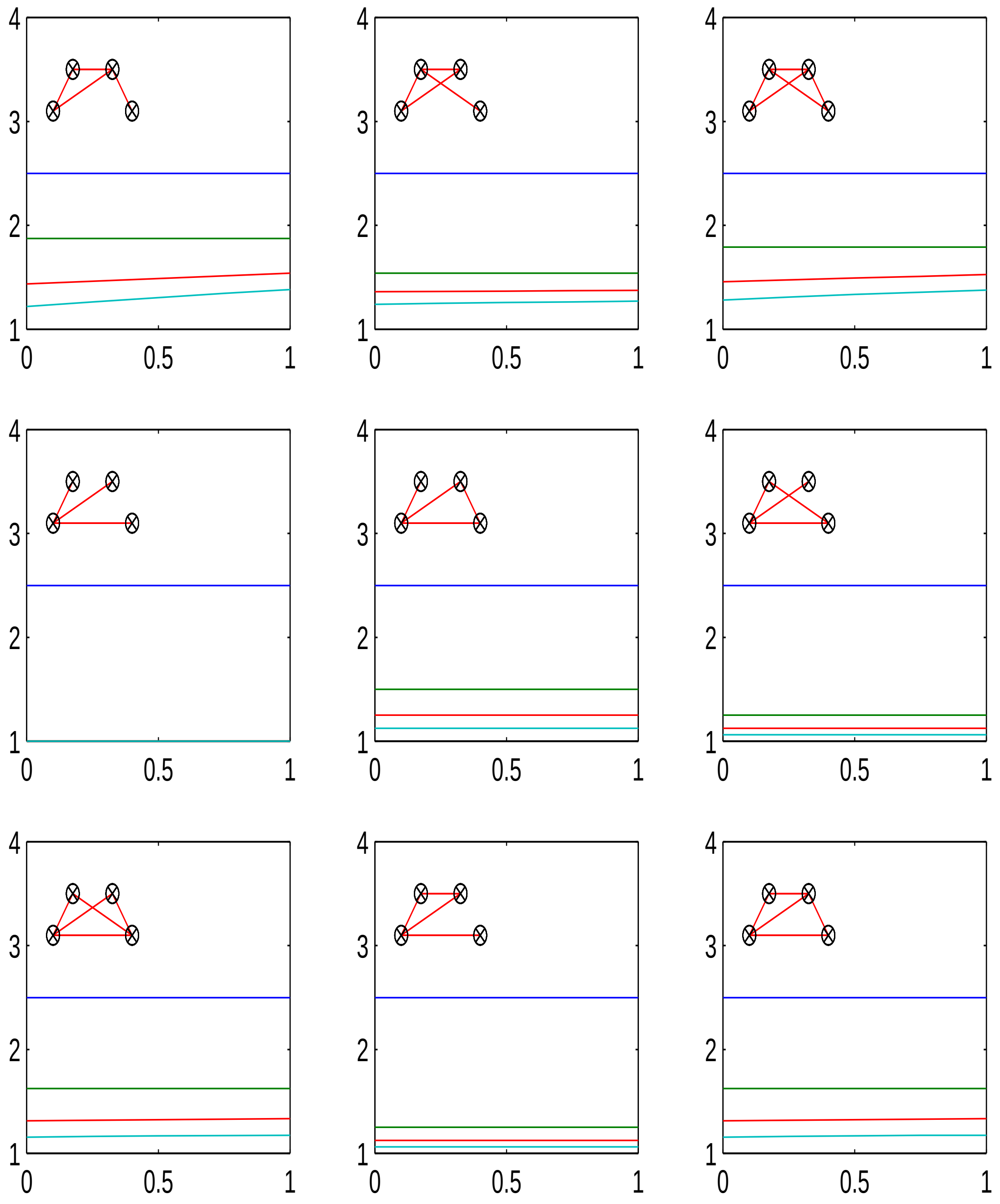


Figure 4.5: Expected Best Cost after 0,1,2,3 Iterations for Random Walk Algorithms for different Neighborhood Definitions (4)

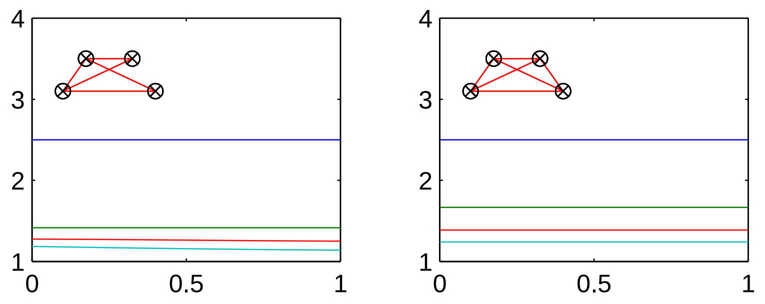


Figure 4.6: Expected Best Cost after 0,1,2,3 Iterations for Random Walk Algorithms for different Neighborhood Definitions (5)

There is another, more trivial example of a neighborhood where this holds. The neighborhood shown in column 1, row 2 of figure 4.5 is such that all points are connected only to $x1$, the global minimum $(N(xi) = x1, \forall i \neq 1)$. This is in some sense, the "perfect" neighborhood – one where the global optimum can be found in 1 iteration at most, no matter what search algorithm is being used. In practice, of course, the neighborhood definitions do not lie at either extremes. Under such circumstances, some algorithms will perform better than others on a given neighborhood.

4.2 The No Free Lunch Theorems for Search

The recently published "No Free Lunch (NFL) Theorems for Optimization" [53], [54] show mathematically that for combinatorial optimization problems the average performance of all algorithms over all possible problems is identical (it is assumed in this analysis that the choice of neighborhood is incorporated into the definition of each algorithm). Therefore, if some algorithm A performs better than algorithm B on some problems (for a given size of search-space), then the reverse must hold for all other optimization problems possible in that search space. In a related paper [55], the authors show that in some sense any problem or class of problems is just as "hard" when averaged over all possible algorithms. These somewhat counter-intuitive result means that it cannot be claimed that any given algorithm will perform better than any other over all possible problems, or that a given problem is "easy" for all algorithms.

In the face of this impossibility of making general claims regarding the performance of algorithms, the performance of each algorithm must be measured for any given problem (or class of problems). For search spaces of non-trivial size, this performance can only be determined experimentally.

4.3 Experimental Analysis – Prior Work

As explained above, a theoretical understanding of the performance of search techniques is extremely difficult due to the complex interactions between the various parameters of the algorithms, the neighborhood definition, and the problem space. Under these circumstances, an experimental analysis of the performance of search algorithms becomes very important. Besides allowing us to determine the suitability of the various techniques for a given problem, such analyses provide insights into the nature of the problem and may even suggest possible improvements to the algorithms.

Though the importance of comparing the performance of various search algorithms is clear, such comparisons appear rarely in the literature. The reasons for this lie chiefly in the fact that most researchers tend to be focused on solving the problem at hand and may not know more than one or two of search algorithms.

In [56], the performance of Greedy Search (with multiple restarts), Tabu Search and Simulated Annealing is experimentally compared for a complex multi-objective flow-shop problem. The authors concluded that as the number of objectives (ad thus the complexity of the problem) increases, Simulated Annealing performs better than Tabu Search. As mentioned before in section 3.2, [32] compared Simulated Annealing with Tabu Search for the Channel Assignment problem, also concluding that SA was better than TS for the problem (though it seems plausible that an inefficient implementation might have been the reason). [33] ran TS on a common test-bed of channel assignment problem instances, where they found their version of TS performed better than or equally well as all previously reported algorithms.

In [57], Genetic Algorithms, Simulated Annealing and Tabu Search are compared for an application in population biology: minimizing the error in the parameterization of a model in metapopulation dynamics. Several variants of GA were tested along with SA, TS for different parameter values. The author observes that a version of GA called "breeder GA (BGA)" produced the most consistent convergence while SA eventually produced the lowest value of the objective function. Simple Genetic Algorithm (with simple mutation and cross-over) and Simulated Annealing are compared in [58] for their performance in the optimal link enhancement problem where the goal is to expand a network by adding new nodes and links in an optimal manner. They found experimentally that while both algorithms are capable of finding the global minimum, they do so in different number of iterations. Specifically, the implementation of SA used was able to find the solution in fewer iterations in all cases.

Experimental comparisons of these algorithms for single machine scheduling problems are given in [59] and [60]. In [59], two versions of greedy search - steepest descent and fastest descent, are compared with Simulated Annealing, Tabu Search, and Genetic Algorithms. Two different neighborhood generating operators were considered for greedy search, SA and TS- move and exchange. For greedy search a combination of the two was found to perform the best and used for comparison, for SA after initial testing only the move operator was used for comparison; Based on similar initial tests the Tabu search used the exchange operator, while for Genetic algorithms the standard binary mutation and uniform crossover operators were considered (as we shall later, this calls into question the fairness of comparisons). Each Algorithm was tried out with various parameters. Based on CPU processing time that it took for the algorithms to converge, the authors conclude that SA performed the best. They however offer two possible explanations for this: (i) SA works well with this search space, (ii) the other algorithms are not implemented as efficiently as they should be (which in the light of the results of [60], appears quite likely). [60] presents a somewhat more rigorous experimental comparison of multi-start and iterated local search algorithms (MLS, ILS) that are essentially repetitions of greedy search once the local optimum is found, GA, SA and TS. It is found that the simple implementation of MLS is competitive to GA, the performance of TS is better than MLS, but Simulated Annealing and a version of GA hybridized with MLS are found to provide the best results overall when longer computational time is provided. These results are also described in [61].

Some other papers describing experimental comparison of algorithms for other combinatorial optimization problems are [62] (quadratic assignment problems), [63] (portfolio optimization in finance), and [64] (cell placement in VLSI design). [65] provides a comprehensive survey of tour construction heuristics, classical local optimization algorithms (2-Opt, 3-Opt, Lin-Kernighan), simulated annealing, tabu search, genetic algorithms, and neural net algorithms.

4.4 Considerations in Performing Experimental Analysis

Performing an experimental comparison of search algorithms for a given problem is not an easy task. A recent paper on the subject, [66], points out some of the numerous pitfalls that a researcher has to avoid while attempting to perform fair, meaningful experimental comparisons. Some of these are:

- Experimentally studying problems of little practical value
- Not referring adequately to the existing literature
- Using inefficient implementations of algorithms
- Not providing explanations and measurements to back them up, leaving obvious anomalies unnoted and unexplained
- Providing non-reproducible results, not providing sufficient details on the implementations
- Using running times as a stopping criterion

- Misusing statistical tools
- Obscuring presentation of data in hard-to-read charts
- Comparing apples and oranges (more on this below)
- Drawing unsupported conclusions from data

At each step in this work, there has been a conscious attempt to avoid such pitfalls. Even so, there are other problems associated with performing experimental comparisons.

Even for a fixed problem and general choice of algorithm, there are numerous algorithm parameters that can be changed which may in conjunction with other parameters affect performance in an unclear fashion. In this thesis, for each algorithm some parameters that appear to be significant are selected and the effect of changes in these parameters on the performance of the algorithms is explored. For example, for Random Walk the value of p the probability of accepting uphill moves is a significant factor. For SA such parameters include the choice of cooling schedule and the rate of cooling. For TS the size of the candidate list and the tabu tenure potentially have an affect performance. For GA the kind of selection used (proportional, tournament, or rank-based selection), the size of the population in each generation are all factors that affect performance. However, one can physically test only a limited number of parameters and their values, and it is hard to test the dependencies between these parameters especially when these are not too obvious. The results of any experimental analysis must be considered valid only for such parameters as were tested, and one must be cautious about making any unsupported extrapolation from such experimental data.

Another major parameter that can be changed is the neighborhood definition. Very often in the literature, these algorithms are compared to each other over *different* neighborhood functions. Typically this is seen when the authors have a favorite algorithm whose effectiveness they wish to demonstrate and the improvements made to the algorithm include changes in how the neighborhood **N** is defined for the search. This is rather like comparing apples and oranges. In figures 4.2 - 4.6 we have seen that, for example, greedy search performs better than Random walk with $p = 0.5$ over some neighborhoods, and vice versa on others. One way of ensuring fair comparisons is to compare these algorithms for the same neighborhood definition[2]. Such a technique is employed in this thesis.

An attempt has also been made to place the work in this thesis in a proper context in the existing literature. Besides providing a general survey of other problems in mobile communications where these algorithms have been applied, papers covering the application of some of these algorithms to the two specific problems considered in this thesis – fixed network optimization [22] and base station location [35], [37], have also been reviewed in Chapter 3.

To maintain fairness in the termination criteria (deciding when an algorithm is to be stopped), we ensure that for each set of comparisons the same number of function evaluations are made. This means for example that a Random Walk or Simulated Annealing run with 100,000 iterations (which corresponds to 100,000 function evaluations) would be compared with a Genetic Algorithm with 2000 generations if the population size is 50 (since the number of function evaluations for the GA would be 50x2000 = 100,000) and a Tabu Search with 10,000 iterations

[2]another option is to compare them for the neighborhoods on which they have been experimentally found to perform the best.

if the size of the candidate list from which the next point is picked is 10 (again corresponding to a total of 10x10000= 100,000 function evaluations).

To obtain experimental data with statistical significance is a desirable goal. All comparisons for the problem of fixed network topology optimization are compared for at least 10 runs, and the minimum, maximum and average final costs found in those runs is shown. Providing such information becomes harder if the problem is computationally complex and it takes several hours for each run to complete. One option is to limit the number of iterations for at least the preliminary parameter-change investigations. This was done for the second problem of optimal base station location.

To ensure reproducibility of results, the source code (in Java) is provided for all the algorithms for both problems in the appendix. Care is taken to ensure that sufficient details about the implementations are provided for anyone who wishes to reproduce these results. An explanation or at least a hypothesis supported by the data is provided in all cases for the results observed.

In some sense the task of an experimental analysis of algorithms is never complete – there is no limit to the number of experiments one can do to test out new ideas and explore the behaviour of the algorithms over new parameter settings, neighborhood definitions or other modifications. One can only report on what has already been accomplished and draw such conclusions from these results as are clearly supported by the data.

Chapter 5

Base Station Location Optimization

5.1 Introduction

As described previously in section 3.3, one optimization problem where search techniques may be utilized is in deciding upon the location of base stations. This is typically done in the planning stage when setting up a new mobile communications network. Information concerning the radio propagation characteristics of the service area, and an initial list of possible sites where base transmitting stations can be located is used to design cells in such a way as to minimize the cost of equipment (i.e. number of base stations used), while maximizing service (i.e. the radio coverage provided in the area). Such multi-objective optimizations involving trade-offs require that some weights be assigned to each objective.

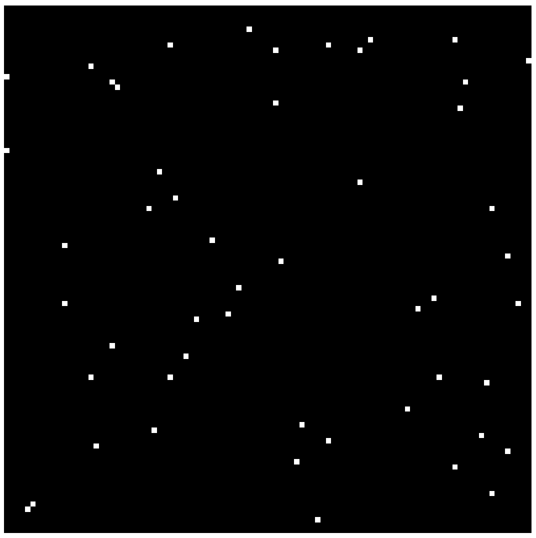


Figure 5.1: Potential Base Station Locations

5.2 Problem Description and Parameters

An experimental comparison of the various search techniques was undertaken to see their relative performance on this problem. A square service area was selected and broken up into a grid of 100x100 points spaced 1 unit distance apart. From this set of 100,000 possible points, 51 locations where base transmitting stations could be potentially be located were generated randomly with a uniform distribution. These 51 possible sites can be seen in Figure 5.1. For real world problems, this initial list of potential base station locations would depend upon geographical considerations, physical constraints and the ease with which radio transmitters can be installed in those locations.

For each potential location a service coverage area was obtained using a log-normal shadow fading model for radio propagation. In this model, the power loss in dB at a distance d from the base station is given by the equation

$$P_{loss}(d) = A + Blog(d) + N \tag{5.1}$$

where N is a zero-mean Gaussian random Variable with a variance σ^2. For this problem the parameters were chosen to be A = 50, B = 40, σ^2 = 10. For each of the possible base station locations, the radio path loss, P_{loss} was computed for all 100,000 points. Then, a cutoff value $P^* = 100dB$ chosen for the path loss, such that if $P_{loss} \geq P^*$ for any point it is deemed that the radio coverage is not sufficient. Figure 5.2 shows the radio coverage for one such transmitter located at one of the 51 possible locations. If all 51 possible locations were selected, the net coverage would be 100% but there would be a significant overlap between different base stations. The overlap factor OF, which represents the average number of base stations providing coverage to each point in the area, was found to be approximately 6.59. This shows clearly that many of the possible locations provide redundant coverage and that far fewer than 51 base stations would be required to provide good coverage in the area.

For this optimization each point in the search space would represent which subset of the 51 potential locations were actually chosen. This can be represented by a 51 bit binary string with each bit corresponding to one of the potential locations. Each bit is one if a base station is placed at the corresponding location and zero otherwise. The size of the search space is therefore $2^{15} \sim 2.25x10^{15}$. The cost function chosen is similar to that used in [35]:

$$f = k\frac{N_{BTS}}{R^{\beta}} \tag{5.2}$$

where N_{BTS} is the number of base transmitting stations selected, R is the radio coverage (percentage of locations in service area which are are covered by at least

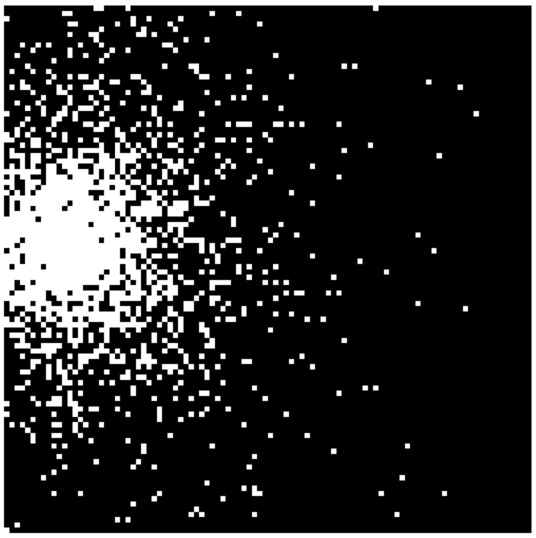


Figure 5.2: Sample Radio Coverage for One Potential Location

one base station) provided by these selected stations, β reflects the weight attached to maximizing coverage as opposed to minimizing the number of base stations, and k is a scaling factor. For this problem the values of the parameters were chosen to be $\beta = 3$, $k = 10^4$.

For this function, a histogram of costs was generated from 10^5 random samples and is seen in Figure 5.3. The histogram shows a long tail, this is explained by the fact that there are a number of possible choices of locations closely spaced such that if they were selected there would be a high number of N_{BTS} that corresponds to low radio coverage. A significant number of search points have cost values between 0.2 and 0.5. The peak around 0.23 corresponds to choices like (19 base stations, 95% coverage) and (16 base stations, 90% coverage). The most optimal points are those with low cost and the lowest values seen in the histogram are about 0.18 which corresponds to solutions like (11 base stations, 85% coverage). It is expected that the search techniques would yield solutions with costs at least

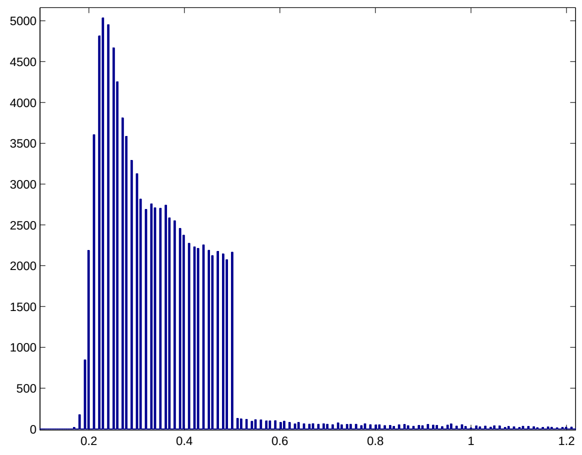


Figure 5.3: Histogram of Costs for Base Station Location Optimization

good as 0.18, if not better. Only then could their performance be said to be better than a random sampling of the search space.

5.3 Neighborhood Definition

Three different Neighborhood definitions were considered for this problem.

- Neighborhood definition A: flip any one bit of the current solution string at random.
- Neighborhood definition B: with a low "mutation" probability P_m (typically about 0.01), flip each bit in the binary string representing the current solution.
- Neighborhood definition C: randomly pick a new point from the entire search space. This is the completely random neighborhood where $N(x) = S$.

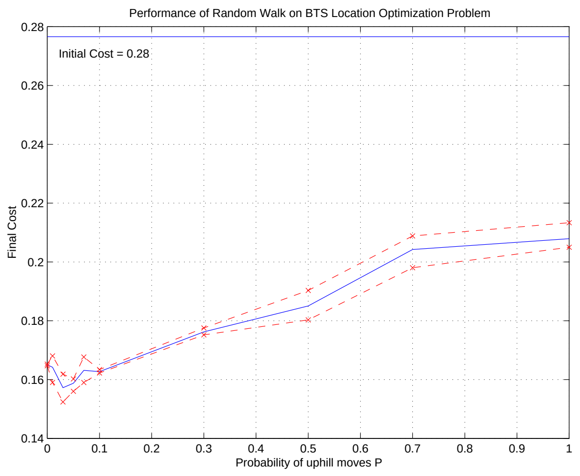


Figure 5.4: Performance of Random Walk Algorithms for Base Station Location Optimization

Since definition A was utilized for most of the experiments performed for this problem, it is assumed in the descriptions below that the neighborhood is defined as per definition A, unless specified otherwise. Also for the experiments described below the initial starting point for the various algorithms was chosen randomly in such a way that the number of initial locations was 26 (approximately half the total number of possible locations).

5.4 Performance of Random Walk

As described in Chapter 2, the chief parameter in the Random Walk algorithms is p, the probability of accepting uphill moves. An experiment was conducted comparing the Random Walks for different values of p. For each p, the Random Walk algorithm was run 10 times for 1000 iterations. Figure 5.4 shows the average final cost obtained in these runs for values of p ranging from 0 to 1. It may be

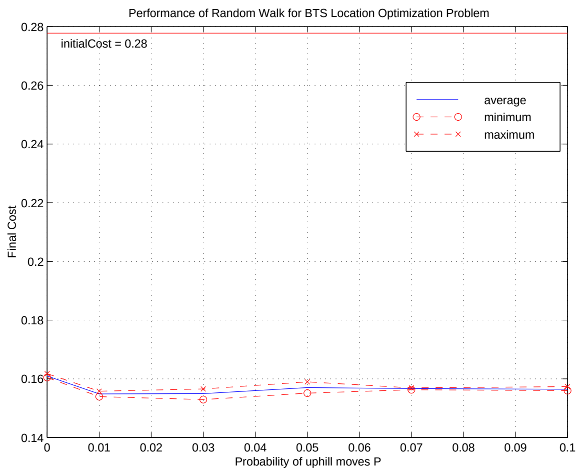


Figure 5.5: Performance of Random Walk Algorithms for Base Station Location Optimization

noticed that the final solutions provided by all Random Walk algorithms are all quite good, and certainly much better than what could be obtained via random sampling of the space. While in general the quality of solutions deteriorates as p increases, there is a small dip at low values of p. To investigate this effect further, the runs were repeated for various values of p ranging from 0 to 0.1. These results are shown in figure 5.5 which plots the average final costs as well as the maximum and minimum values observed in the runs. It is seen that for low non-zero values of p, the algorithms provide better final solutions than $p = 0$ which is a greedy search. Random Walk with $p = 0.03$ was found to provide the best overall solution.

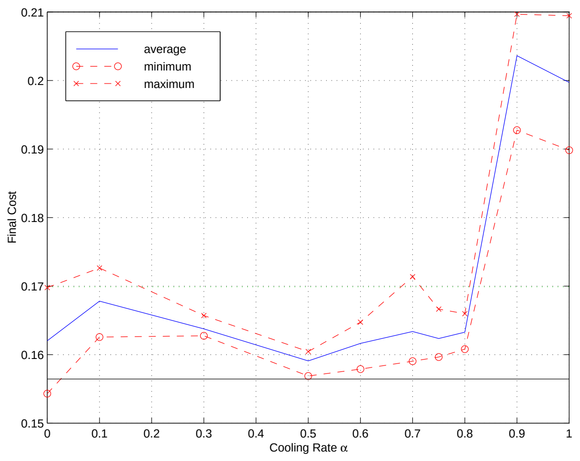


Figure 5.6: Performance of Simulated Annealing for Base Station Location Optimization

5.5 Performance of Simulated Annealing

If I is total number of iterations to run the algorithm and n the number of discrete temperature levels in each run, the length L of the homogeneous Markov chains was set to be

$$L = \frac{I}{n} \tag{5.3}$$

For this problem n was chosen to be 20 (i.e. 20 different temperature values are selected, and if $I = 10^4$ then $L = 500$). The initial temperature T_o needs to be high enough. It was determined by first obtaining L cost values via a purely random search (RW with $p = 1$) and finding the standard deviation σ_∞ of these L costs and setting $T_o = 10\sigma_\infty$. The cooling schedule was chosen to be geometric:

$$T_{new} = \alpha_c T_{old} \tag{5.4}$$

where α_c is the cooling rate.

If the cooling rate α_c is too low, then the Simulated Annealing process would go down to zero temperature quickly and perform a greedy search. If, on the other hand, α_c is too high, the algorithm would not cool quickly enough and its behaviour would be more like a Random Walk algorithm with $p = 1$. To identify the optimum value of α_c the algorithm was compared for values ranging from 0 to 1 (10 runs for each values). The number of iterations I for this comparison was chosen to be 1000, hence the lengths of the isothermal Markov chains $L = 50$. All runs are compared for the same initial point in the search space. The results are shown in figure 5.6 which shows the average, minimum and maximum final costs for each value of α_c. For comparison, the average performance of greedy RW with $p = 0$ (dark dotted line with cost 0.17), and of RW with $p = 0.03$ (plain line with cost 0.157) are also shown. As expected, for low values of α_c, SA performs about as well as (but no worse than) greedy search. For high values of α_c (greater than 0.8), the performance deteriorates significantly. The optimum value for which SA performs well is found to be $\alpha_c \sim 0.5$. However, it does not appear that SA performs any better than RW with $p = 0.03$.

5.6 Performance of Tabu Search

For this problem the definition of which moves are considered Tabu is quite simple. Those base station locations that were selected within the last K iterations are considered Tabu, where K is the *Tabu Tenure*. Besides the value of K, another important parameter in Tabu Search is the branching factor at each step - the size of the Candidate List v, which determines how many neighbors are considered at each step in the search.

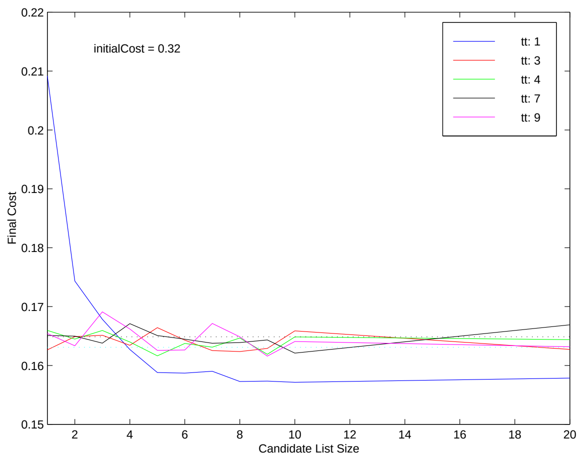


Figure 5.7: Performance of Tabu Search for Base Station Location Optimization

To determine the effect of the values of K and v on the performance of Tabu Search, the algorithms were compared for different parameter values. This was done for the same initial starting point for 10 runs each consisting of 1000 function evaluations (the number of iterations $I = \frac{1000}{v}$). The algorithms were compared for $K = [1, 3, 4, 7, 9]$ and $v = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 20]$. One would expect that if the Tabu Tenure K is high, given the above definition of tabu moves, the performance would deteriorate. This is because if the tenure is high, then if the new Base Station located at some iteration in the search is a good one, it won't be considered for the next K generated solutions (which may not be as good). In terms of the candidate list size v, one might expect that there would some improvement as the size of the candidate list increases since one can better sample the neighborhood of the current point in order to determine a good point to move to.

The results of the experiments are shown in Figure 5.7. It is observed that indeed Tabu Search performs best for the lowest possible value of the tabu tenure

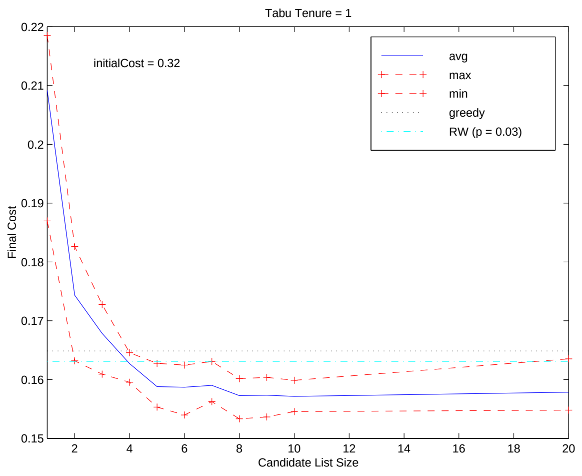


Figure 5.8: Performance of Tabu Search for Base Station Location Optimization $(K = 1)$

$(K = 1)$, and the performance with higher values of K is somewhere between the performance of greedy search and RW with $p = 0.03$ (the two dashed lines) which appear to be stuck in relatively poorer regions in the search space. Figure 5.8 shows the performance for this tenure with additional details on the minimum and maximum final cost values observed in the 10 runs. Further, for the case where $K = 1$, Tabu Search is seen to perform progressively better as v is increased, reaching the best performance for a candidate list size of 10.

5.7 Performance of Genetic Algorithms

Given the neighborhood definition A (described in section 5.3, the parameters for Genetic Algorithms that can affect performance are the type of selection used and the size of the populations in each generation. Three types of selection methods were compared with each other. These selection techniques were implemented as described in [67], [68], [69]. These are:

- Proportional Selection: At each generation the probability of selecting an individual chromosome in the population for reproduction is proportional to its fitness value. One problem with this selection technique is that it is not invariant with respect to the scaling of the cost-functions which means that the fitness function must be a scaled inverse of the cost function. Exponential scaling was utilized with an exponent of 100.
- Tournament Selection: A tournament is conducted by picking t chromosomes in the population at a time randomly and selecting the best chromosome among those for reproduction. This tournament is repeated until the next generation can be created. The tournament size t was fixed to be 2 (*Binary Tournament Selection*). This selection scheme is scaling invariant and requires no scaling.
- Rank-based Selection: In this all the chromosomes in a population are ranked according to their fitness and the probability of selecting a chromosome is made proportional to its rank. One parameter for Rank-based selection is β_{rank} which is defined to be the expected number of offspring of the best chromosome in the population. For the experiments here, β_{rank} was assigned the value 1.5 which is a typical value. One potential problem with Rank-based selection is that in some sense the exact value of the cost functions for each solution is ignored. While this is useful for objective functions where the exact cost values are not easy to obtain, it could result in sub-optimal performance when the exact cost values are known.

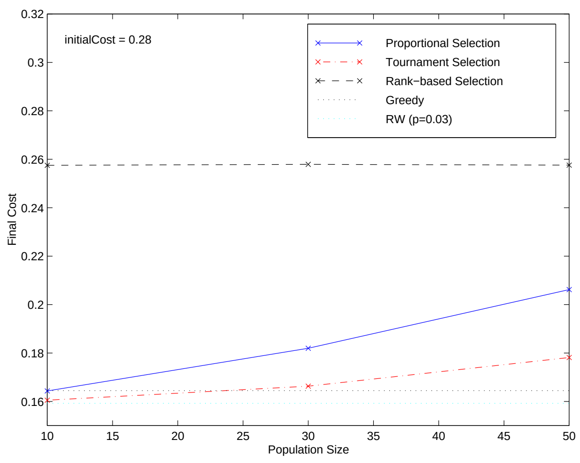


Figure 5.9: Performance of Genetic Algorithms for Base Station Location Optimization

Another factor that can influence the performance of Genetic Algorithms is the size of the population. Figure 5.9 shows a comparison of the average performance Genetic Algorithms for the three different selection techniques and for population size values of $PS = [10,\ 30,\ 50]$. All runs were conducted for 1000 cost function evaluations. It is observed that Rank-based Selection performs quite poorly (and performs the same for all three population sizes tested), while Tournament Selection performs the best of the three selection techniques. With both Tournament and Proportional Selection the performance seems to deteriorate linearly with increasing population size with the best performance at a value of 10. One explanation for this is that since we are comparing all the algorithms for the same number of cost evaluations (1000), the number of generations $G = \frac{1000}{PS}$ is smaller when the size of the population is higher. This suggests that the performance of the Genetic Algorithms improves linearly with the number of generations for these two selection schemes, at least for the first 1000 function evaluations.

5.8 Comparison of Algorithms

The different search techniques were first compared with each other for their performance on the problem using the best parameters from the previous experiments. Figure 5.10 is a bar chart comparing the performance of Random Walk with $p = 0$ (Greedy Search), Random Walk with $p = 0.03$, Simulated Annealing, Tabu Search, and Genetic Algorithms. These algorithms are compared for Neighborhood definition A (the default neighborhood), and each was run 10 times for 10,000 function evaluations. The figure shows the maximum, average and minimum final cost values obtained by each search technique. We observe first that all techniques provide roughly the same performance though the results of Greedy Search show a high variance indicating that it has a tendency to get trapped in local minima. Also the minimum final values obtained by all of them in 10 runs are nearly identical. The performance of Simulated Annealing both in terms of variance and mean is better than Greedy Search but slightly worse than Random Walk with $p = 0.03$, which in turn appears slightly worse Genetic Algorithm. The performance of Tabu Search is observed to be the best of them all, both in terms of average final cost and variance.

It was observed that the Random Walk Algorithms (both for Greedy Search and for $p = 0.03$) tended to settle down to solutions quite early in the search and not move from there. Since this is wasteful of function evaluation calls, one approach is to re-start this algorithm from a randomly generated point in the search space. This was done for RW, modifying them so that they re-start if no change in the current point is seen for more than 50 iterations. The multi-start RW algorithms were then compared with SA, TS and GA (again for 10 runs). This is seen in

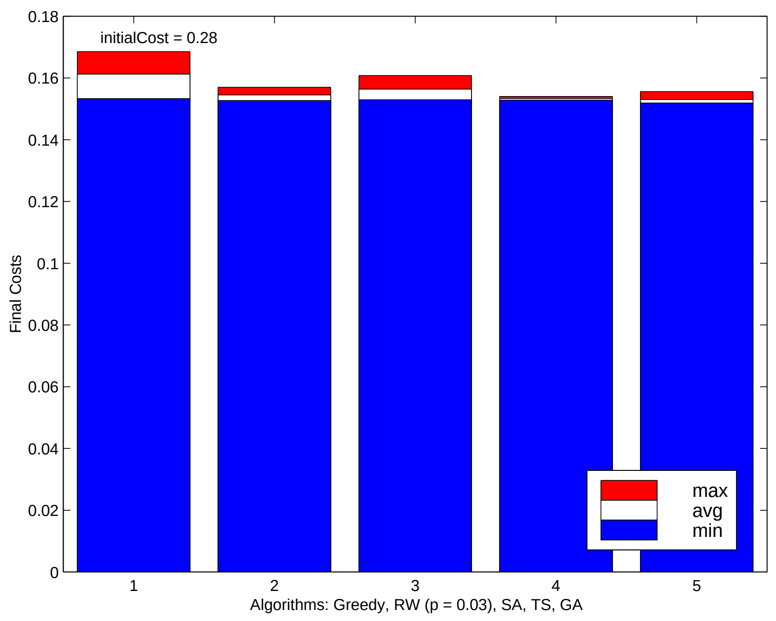


Figure 5.10: Comparison of Search Algorithms for Base Station Location Optimization

Figure 5.11. This clearly improves the performance of Greedy Search (even to the point of a small improvement over RW with $p = 0.03$) which is now comparable to the performance with GA and better than SA. Tabu Search, however, still offers the best performance. Figure 5.12 shows one sample run of each algorithm comparing the best cost to date vs. number of function evaluations. In this run, TS and GA provide the best final cost while all the other converge to about the same final cost, which is still quite close to that obtained by TS.

5.9 Effect of Neighborhood Definition

As described before in chapter 4, the definition of neighborhood used in the search procedure can affect performance. To investigate this further, experiments were

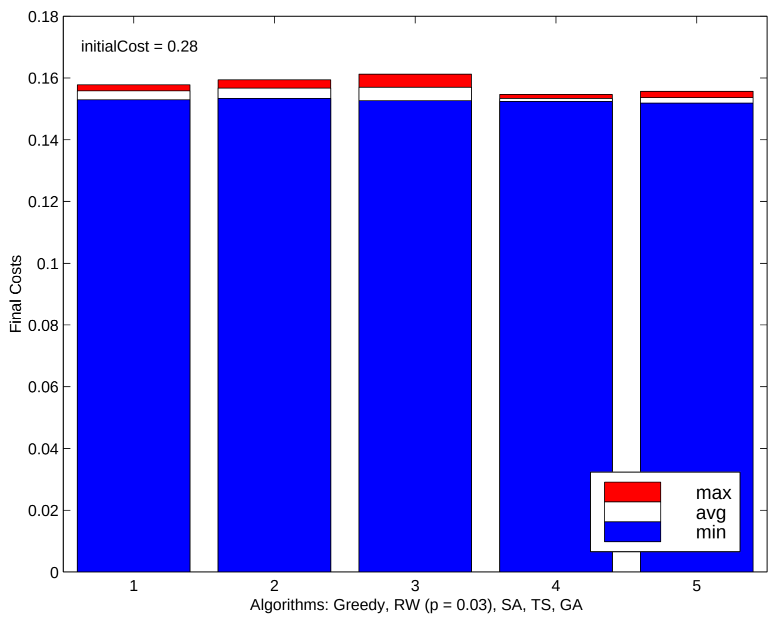


Figure 5.11: Comparison of Search Algorithms for Base Station Location Optimization (with Multi-start Random Walk)

performed to compare the performance of each search technique for other neighborhood definitions. Figures 5.13 - 5.15 show the results of utilizing neighborhood definition B (see section 5.3) with varying values of P_m the probability of "mutating" or flipping each bit. The three figures are for P_m = 0.001, 0.01 and 0.1 respectively. Increasing the probability of mutation makes the neighborhood more random. When $P_m = 0.1$, for example, there is a one in a ten probability of changing each bit in one step which means that neighboring points will on average differ by about 5 bits (0.1 x 51). For each P_m the algorithms were run 10 times for 10,000 function evaluations.

The results are interesting. For $P_m = 0.001$, as seen in figure 5.13, we find that while Tabu Search still provides the best overall results, the performance of Simulated has been improved considerably, making it perform even better than

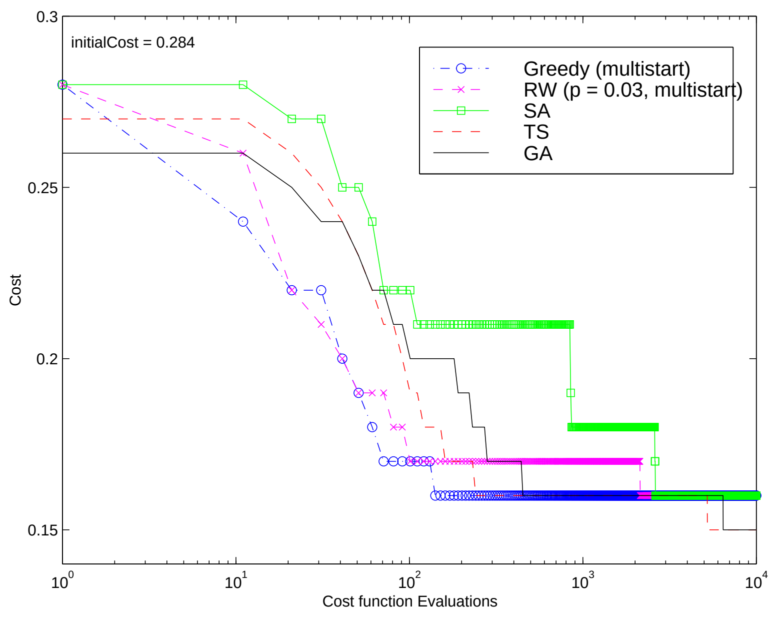


Figure 5.12: Comparison of Search Algorithms for Base Station Location Optimization (1 Sample Run)

GA. The Random Walk algorithms are seen to perform rather poorly. This is perhaps due to the restrictive nature of the neighborhood definition - for this low value of P_m, only one in two neighbors will differ by a bit. This means there is a greater chance for the Random Walk algorithms to be stuck in a small portion of the search space and provide locally optimal solutions. Because the Simulated Annealing algorithm goes through a random walk when the temperature is high initially, it is able to locate good regions of the space which it explores more greedily as the temperature cools down to zero. Tabu search is still able to perform well because the tabu list directs it out of poor regions. The Random Walk algorithms perform better as P_m is increased, and as seen in figures 5.13, 5.13, their performance is comparable to the other algorithms. Relative to the other algorithms, Simulated Annealing performs much better at $P_m = 0.1$, providing the lowest average final cost. Another result observed is that Tabu Search provides a slightly worse final cost for this value of P_m but in the ten runs observed, the final cost obtained shows little or no variance. These results demonstrate clearly the dependence of the performance of search algorithms on the definition of the neighborhood for a given problem.

In chapter 4, we argued that when the neighborhood is completely random i.e. $N(x) = S, \forall x \in S$, all algorithms would provide the same performance. This effect was verified by utilizing Neighborhood C (described in section 5.3). Figure 5.16 shows the result of comparing the search algorithms for 10,000 function evaluations for 10 runs each. As predicted the performance is nearly identical for all algorithms for this neighborhood. It is also worth noting that the final cost in all runs was about 0.27, significantly worse than the performance for more structured, localized, neighborhood schemes. This shows that search algorithms with reasonable neighborhood schemes do provide much better results than randomly sampling the problem space.

5.10 Final Solution

The best overall solution in these experiments was found using Tabu Search (with tabu tenure 1 the size of candidate list 10) and Neighborhood type A. The final cost of this solution was 0.153 and it corresponds to 10 Base stations offering 86.82% coverage of the area. This near-optimal solution is shown in figure 5.17. The crosses (in red) indicate the positions of the final base stations. Points that are not covered are indicated by the black areas, while for the other covered areas,

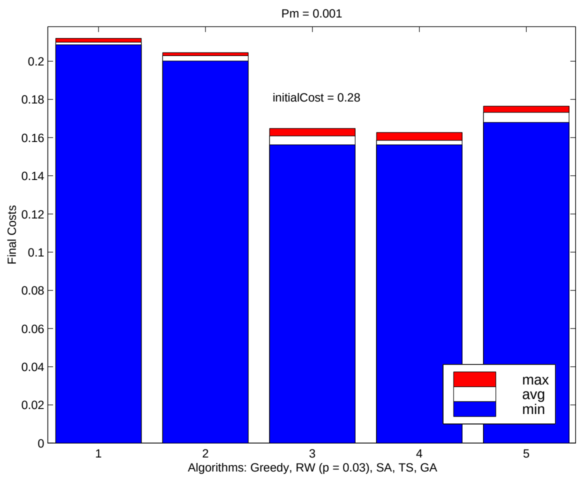


Figure 5.13: Comparison of Search Algorithms for Base Station Location Optimization (Neighborhood B, $P_m = 0.001$)

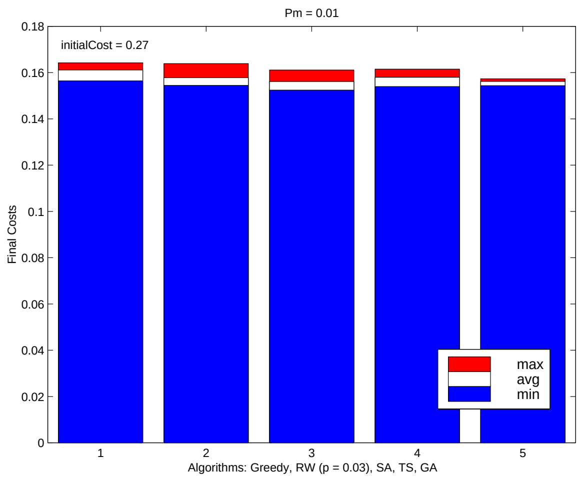


Figure 5.14: Comparison of Search Algorithms for Base Station Location Optimization (Neighborhood B, $P_m = 0.01$)

the coverage is indicated by colors ranging from blue (points covered by 1 BS) to yellow (points covered by 5 overlapping BS).

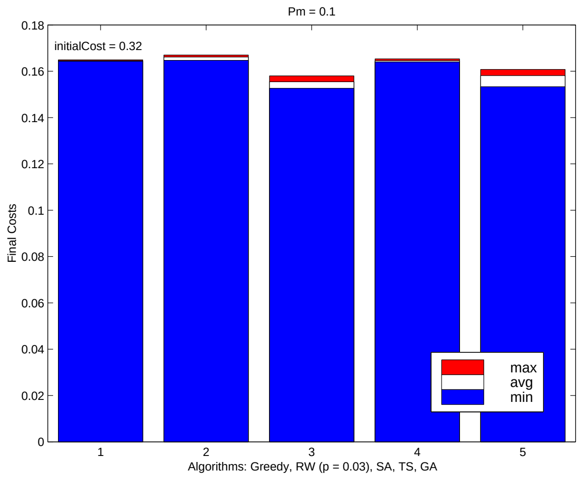


Figure 5.15: Comparison of Search Algorithms for Base Station Location Optimization (Neighborhood B, $P_m = 0.1$)

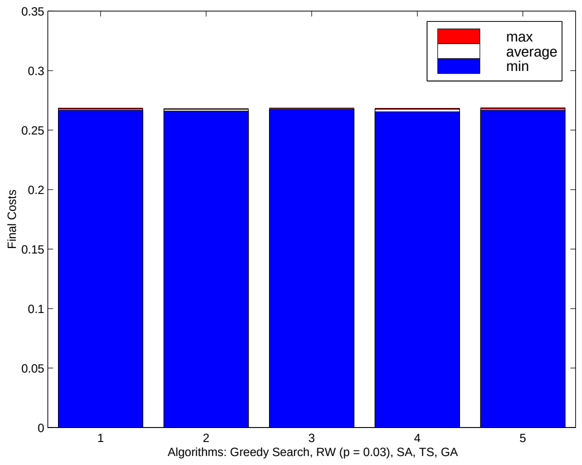


Figure 5.16: Comparison of Search Algorithms for Base Station Location Optimization (Neighborhood C)

Figure 5.17: Best Final Solution for Base Station Location Problem

Chapter 6

Fixed Network Topology Optimization

6.1 Introduction

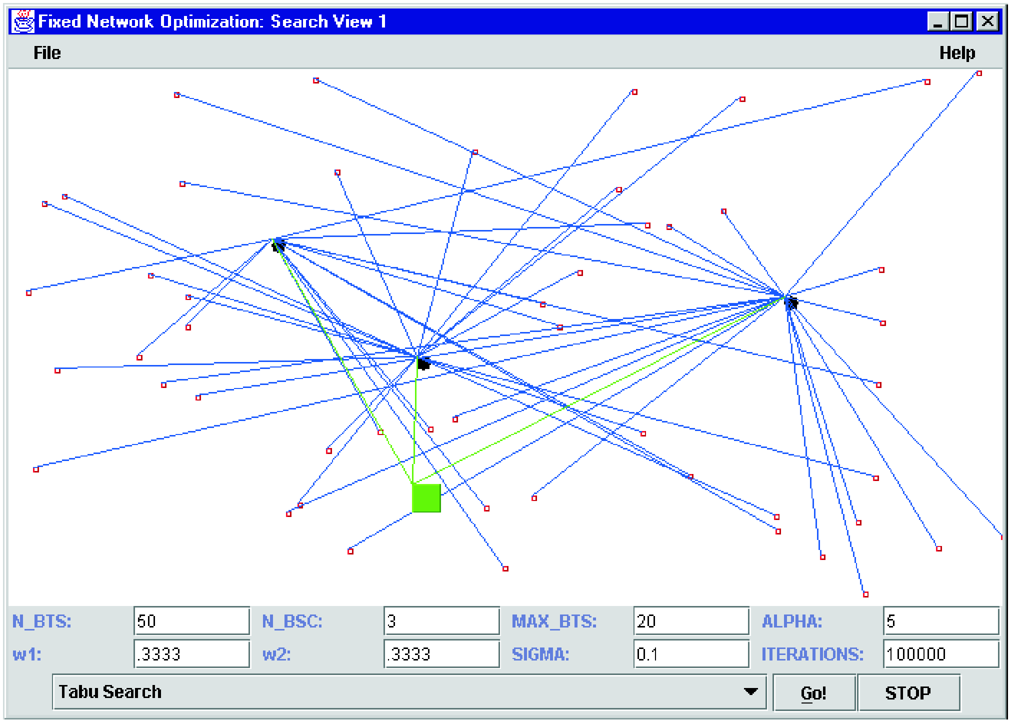
Another problem where the usage of search techniques is desirable is in the topological design of the fixed (wired) portion of the mobile communications system. This problem was described in section 3.1. The cost of the topology of a fixed network depends upon the cost of the nodes, the cost of links, and constraints such as the maximum number of links allowed per node. Both Genetic Algorithms and Simulated Annealing have been utilized to find low-cost topologies [22], [23] for GSM and DCS1800 systems. An experimental comparison of these algorithms and other search technique has not yet been done for this problem.

6.2 Problem Description and Parameters

The network to be minimized consists of N_{BTS} base transmitting stations (BTS) which are connected to N_{BSC} base station controllers (BSC) in such a way that no BTS is connected to more than one BSC, and no more than MAX_BTS of them may be connected to each BSC. It is further assumed that all BSCs are connected to 1 mobile switching center (MSC). The locations of all base stations is pre-specified, as is the location of the MSC. The object of the optimization process is to find the best location and allocated BTSs for each of the N_{BSC} BSCs. In order to permit comprehensive experimental analysis and in the absence of industrial data, some simplifying assumptions were made in the problem model. It is assumed that N_{BSC} is specified based on cost of equipment considerations. Since the cost of nodes is thus fixed, the cost of the network topology depends solely on the length of the various links. It is assumed that the cost per unit length of the higher bandwidth links between BSCs and MSC when compared to the cost per unit length of the links between the BSCs and the BTSs is greater by a factor $\alpha \geq 1$.

Unless otherwise specified below, all experiments were performed under the assumption that $N_BSC = 3$, $N_BTS = 50$, $MAX\_BTS = 20$ and $\alpha = 5$. These parameters are only varied in the experiments that investigate the effect of problem scaling and different relative costs on the performance of algorithms. It is assumed that the network is located in an area that is of size 10 units x 10 units. For each experiment, the locations of the various BTS were generated randomly using a uniform distribution over the service area and a random assignment of BTSs was chosen. A typical starting point in the search is shown in figure 6.1.

The rectangle represents the MSC, the BSCs are represented by filled circles, and the BTS are represented by the small squares.



N_BTS: 50
N_BSC: 3
MAX_BTS: 20
ALPHA: 5
w1: .3333
w2: .3333
SIGMA: 0.1
ITERATIONS: 100000
Tabu Search

Figure 6.1: Typical Initial Point for Fixed Network Topology Minimization

The cost function to be minimized is:

$$f = \sum_{i=1}^{N_{BTS}} l_{BTS_i \rightarrow BSC(i)} + \alpha \sum_{j=1}^{N_{BSC}} l_{BSC_j \rightarrow MSC} \tag{6.1}$$

where $l_{BTS_i \rightarrow BSC(i)}$ is the length of the direct link between the i^{th} BTS and the BSC it is assigned to (subject to the constraint that no more than MAX_BTS BTSs are assigned to each BSC, and $l_{BSC_j \rightarrow MSC}$ is the length of the direct link between the j^{th} BSC and the MSC.

Figure 6.2 shows the histogram of costs obtained by taking 10^6 samples from a search space with $N_{BSC} = 3$, $N_{BTS} = 50$, $MAX\_BTS = 20$, and $\alpha = 5$. It is interesting to note that for this problem the distribution of costs is quite smooth

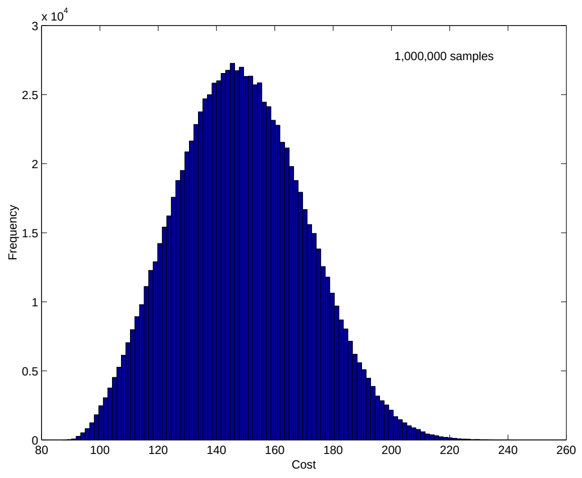


Figure 6.2: Histogram of Costs for Fixed Network Topology Minimization Problem

and appears almost Gaussian (which it is not, of course, since the tails are not of infinite length and there is a slight asymmetry between the left and right halves). The probability of locating low cost solutions falls off quite sharply.

6.3 Neighborhood Definition

Each point in the search space is a particular set of locations for each BSC, and a particular assignment of BTSs to each BSC satisfying the MAX_BTS constraint. To generate a new neighbor from this point two neighborhood generating operators are required, one that moves the locations of the BSCs and another that changes the BTS assignments for each BSC.

To move the BSCs two independent, Gaussian random variables are generated N_x and N_y, both of which are zero mean with standard deviation σ. At each step, one BSC is randomly picked and if it is located at the co-ordinates (X_j, Y_j), in the generated neighbor this BSC is displaced by (N_x, N_y) and has co-ordinates

$$(X'_j, Y'_j) = (X_j + N_x, Y_j + N_y) \tag{6.2}$$

To ensure that the generated BSC location is within the boundaries of the service area, the following hold:

$$X_j + N_x > 10 \Rightarrow X'_j = 10 \tag{6.3}$$

$$X_j + N_x < 0 \Rightarrow X'_j = 0 \tag{6.4}$$

$$Y_j + N_y > 10 \Rightarrow Y'_j = 10 \tag{6.5}$$

$$Y_j + N_y < 0 \Rightarrow Y'_j = 0 \tag{6.6}$$

To modify the assignments of the various BTS to each BSC, three kinds of moves are possible:

- BTS move A: Do nothing. This "null" move is important to have because it allows the search to optimize the location of the BSCs without changing BTS assignments.
- BTS move B: Pick two BSCs, BSC_1 and BSC_2, and if the number of BTSs allocated to BSC_2 is less than MAX_BTS, pick one BTS assigned to BSC_1 at random and assign it to BSC_2 instead.
- BTS move C: Pick two BSCs, exchange one BTS assigned to each (picked randomly with uniform probability) with the other.

Weights may be assigned to each of these kinds of moves. This is accomplished by using two probabilistic parameters ω_1, ω_2 which are chosen such that

$$0 \leq \omega_1 + \omega_2 \leq 1 \tag{6.7}$$

At each step, move A is carried out with probability ω_1, move B with probability ω_2 and move C with probability $1 - (\omega_1 + \omega_2)$.

Thus the parameters that affect the definition of the neighborhood for the various search algorithms are σ - the standard deviation of the BSC displacements, and ω_1, ω_2, which control the changes in BTS assignments to each BSC. Unless otherwise specified (as in the experiments demonstrating the effect of Neighborhood definition on the performance of the search algorithms), it is assumed that $\sigma = 0.1$, $\omega_1 = \omega_2 = \frac{1}{3}$.

6.4 Performance of Random Walk

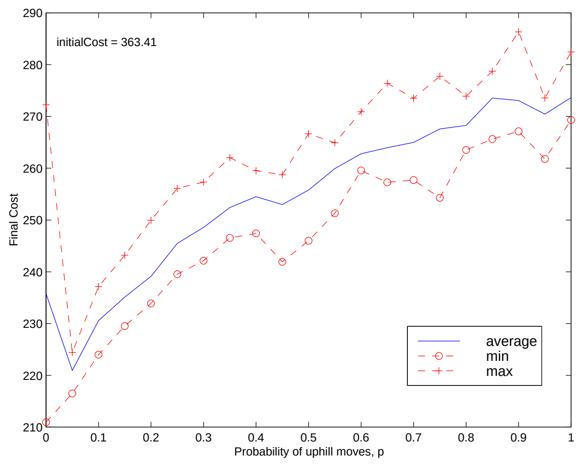


Figure 6.3: Performance of Random Walk Algorithms for Fixed Network Topology Optimization

To investigate the performance of the RW algorithms for varying values of p, the probability of accepting uphill moves, the algorithms were run 10 times for 100,000 function evaluations for each value of p ranging from 0 to 1 in increments of 0.05. The results are shown in figure 6.3. It is observed that the variance of final solutions obtained is greatest with Greedy Search, however the average final cost is worse than that observed with a low uphill move acceptance probability of 0.05. A small value of p dislodges the search from some local minima; however, as p is increased further, the search becomes increasingly more random and the performance deteriorates in a monotonic fashion, with the worst performance observed when $p = 1$ (purely random movement). This is result is analogous to that observed for the base station location optimization discussed in the previous chapter and is fairly typical of search spaces with some local optima (since in the absence of such local optima, greedy search would provide the best performance).

6.5 Performance of Simulated Annealing

If I is total number of iterations to run the algorithm and n the number of discrete temperature levels in each run, the length L of the homogeneous Markov chains was set to be

$$L = \frac{I}{n} \tag{6.8}$$

For this problem n was chosen to be 20 (i.e. 20 different temperature values are selected, and if $I = 10^4$ then $L = 500$). The initial temperature T_o needs to be high enough. It was determined by first obtaining L cost values via a purely random search (equivalent to RW with $p = 1$) and finding the standard deviation σ_∞ of costs at this temperature and setting $T_o = 10\sigma_\infty$. Three cooling schedules were considered:

- Geometric Cooling Schedule: This is the simple cooling schedule,

$$T_{new} = \alpha_c T_{old} \tag{6.9}$$

where α_c is the cooling rate parameter.

- Logarithmic Cooling Schedule: In this cooling schedule,

$$T_{new} = \frac{T_{old}}{1 + \frac{T_{old}\log(1+\delta)}{3\sigma_{old}}} \tag{6.10}$$

where δ is a distance parameter, and σ_{old} the standard deviation of costs at T_{old}.

- Exponential Cooling Schedule: In this schedule,

$$T_{new} = T_{old}\exp(\frac{-\delta_e T_{old}}{\sigma_{old}}) \tag{6.11}$$

where δ_e is the exponential cooling parameter, and σ_{old} the standard deviation of costs at T_{old}.

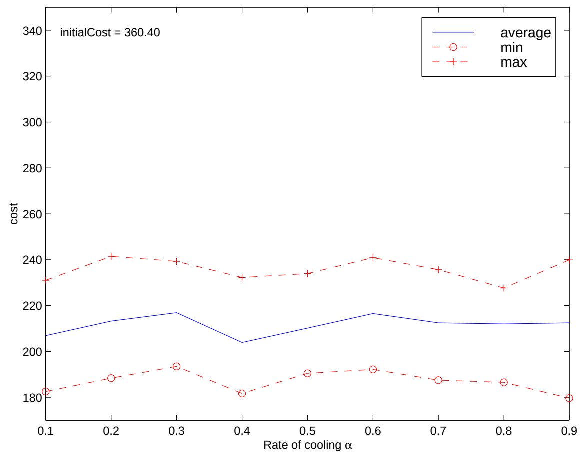


Figure 6.4: Performance of Simulated Annealing with Geometric Cooling for Fixed Network Topology Optimization (1)

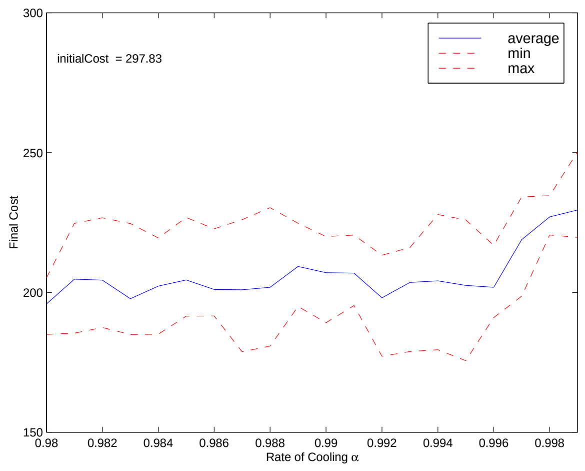


Figure 6.5: Performance of Simulated Annealing with Geometric Cooling for Fixed Network Topology Optimization (2)

To the investigate the performance of the geometric cooling schedule, SA with different values of α_c (which must be between 0 and 1) were each run for 10 runs and 100,000 iterations. Figures 6.4 and 6.5 show the final costs obtained for α_c in the ranges [0.1, 0.9] and [0.980, 1.000] respectively. It was observed that SA performance i the same for all values of α_c from 0 to about 0.996, after which it is seen to deteriorate a little as it is increased to 1.0 (when the cooling is too slow). Further, a visual examination of the lowest cost solutions obtained by SA showed that they were far from optimal. This behavior of SA suggests that the search space may be quite rough and full of local minima. This means that during the initial stages of the annealing process, when the temperature is high, the algorithm is unable to locate good regions of the space to settle down to. If the cooling rate is fast enough to allow the SA to settle down to a greedy search, the net performance of the algorithms depends highly on the greedy portion of its search, which, because of the existence of multiple local minima fails to find good solutions.

The logarithmic cooling schedule was explored next. In general, using various cooling schedules can provide different performance results. But as we have seen, SA seems to perform poorly on this search space for nearly all geometric cooling rates (fast or slow). It seems therefore unlikely that it would perform dramatically better for other cooling schedules. This was confirmed by running the algorithm with a logarithmic cooling schedule for various values of the distance parameter δ ranging from 0.1 to 0.9, as seen in figure 6.6. The minimum, maximum and average final costs found in 10 runs (of 100,000 iterations each) for the logarithmic cooling schedule are compared with the average results for geometric cooling with $\alpha_c = 0.99$. It is seen clearly that the average performance is about the same for the two kinds of cooling schedules.

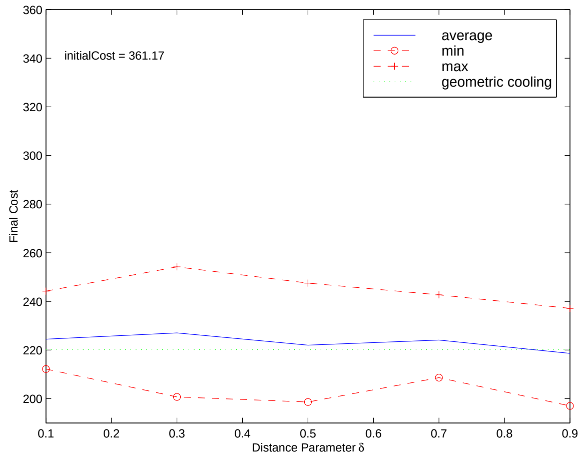


Figure 6.6: Performance of Simulated Annealing with Logarithmic Cooling for Fixed Network Topology Optimization

Figure 6.7 shows the minimum, maximum and average final costs found in 10 runs (of 100,000 iterations each) via an exponential cooling schedule for values of the exponential cooling rate parameter δ_e ranging from 0.1 to 0.9. While at first glance it may appear that as the δ_e is increased, the performance gets better, this is actually not the case for the following reason. As δ_e is increased, the cooling becomes faster. It was observed that for values of δ_e greater than about 0.2, the annealing process spends a significant proportion of time in the zero temperature state, i.e. it performs a primarily greedy search. The high variance observed for greedy search, which is because of its high sensitivity to starting points, can result in performance trends (in this case the trend of decreasing final costs as δ_e approaches 0.9) that are not of statistical significance [1]. To verify this, SA was tested further for δ_e ranging from 0.9 to 1.0, where it was again compared to geometric cooling. Figure 6.8 makes it clear that the performance of exponential cooling for high values of δ_e is no better than the results obtained with geometric cooling ($\alpha_c = 0.99$). Figure 6.9 shows the performance of exponentially cooled SA for low cooling rates (δ_e ranging from 0.01 to 0.1) where again it is clear that the performance of SA is not significantly affected by the choice of cooling schedule.

6.6 Performance of Tabu Search

As there are essentially two kinds of moves that generate neighboring points (moving the locations of BSCs and changing the BTSs allocated to BSCs), two different tabu lists were implemented for the Tabu Search algorithm, both with the same tabu tenure value K. These tabu lists were defined as follows:

- BSC Location Tabu: When the j^{th} BSC is moved to a location (X_j, Y_j) in

[1]This illustrates quite clearly how unsupported conclusions may be drawn from raw experimental data in some cases if the results aren't interpreted in the right context.

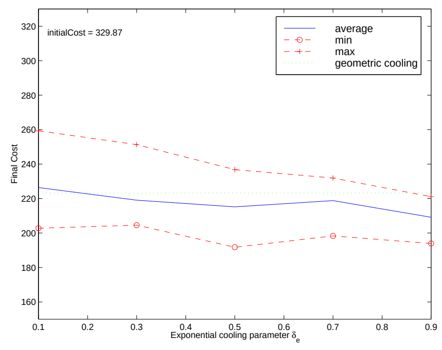


Figure 6.7: Performance of Simulated Annealing with Exponential Cooling for Fixed Network Topology Optimization (1)

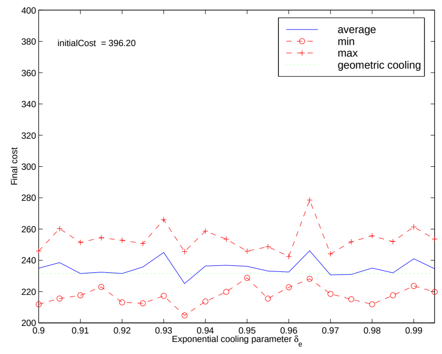


Figure 6.8: Performance of Simulated Annealing with Exponential Cooling for Fixed Network Topology Optimization (2)

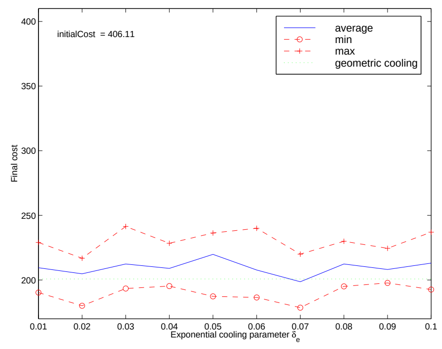


Figure 6.9: Performance of Simulated Annealing with Exponential Cooling for Fixed Network Topology Optimization (3)

iteration n, movement of the j^{th} BSC to any point within a radius of 0.1σ of (X_j, Y_j) before iteration $(n+K)$ is tabu.

- BTS Allocation Tabu: When the i^{th} BTS is taken from the j^{th} BSC and allocated to some other BSC in iteration n, the re-assignment of the i^{th} BTS to the j^{th} BSC is tabu until iteration $(n+K)$.

The simple aspiration criteria , whereby a tabu move is accepted if it leads to the lowest cost function evaluation to date, is utilized in conjunction with these tabu definitions.

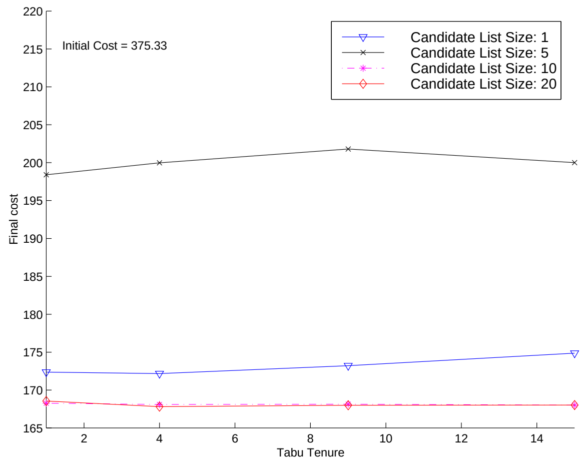


Figure 6.10: Performance of Tabu Search for Fixed Network Topology Optimization (1)

The effect of the value of the tabu tenure K and the size of the candidate list v on the performance of Tabu Search for this problem was investigated. The first experiment carried out analyzed the average final cost over 10 runs of 100,000 iterations each obtained by running Tabu Search algorithms with candidate list sizes $v = [1, 5, 10, 20]$ for values of the tabu tenure $K = [1, 4, 9, 15]$. The results are shown in figure 6.10 and are somewhat intriguing. First of all, it appears that the performance for the various values of K is nearly identical suggesting that in this case the tenure value of 1 may be sufficient to prevent cycling. Further, while the performance for candidate list size $v = 1$ is fairly good, the performance for $v = 5$ is worse, and the final costs obtained with $v = 10$ and $v = 20$ are nearly identical and better than that obtained with $v = 1$, suggesting that a candidate list size of $v = 10$ is good. Since the algorithm appears fairly independent of the value of K, the variation of the final costs with v was further explored by an experiment where K is fixed at a value of 4. Figure 6.11 shows the average, minimum and maximum cost values obtained over 10 runs of 100,000 iterations of Tabu Search for v values from 1 to 10. The average results obtained with the greedy search algorithm (RW with $p = 0$) are also plotted for comparison.

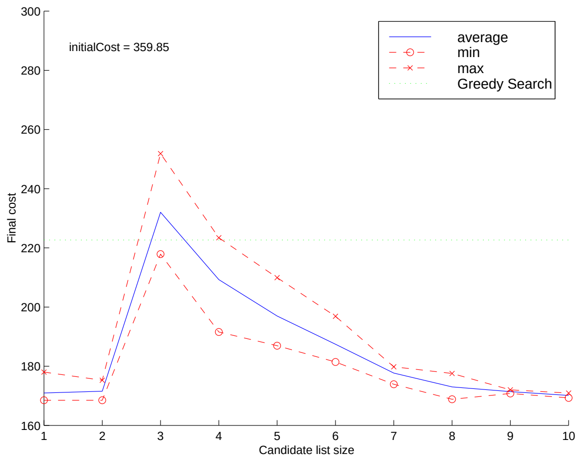


Figure 6.11: Performance of Tabu Search for Fixed Network Topology Optimization (2)

It appears from figure 6.11 that the performance of Tabu Search is fairly good for candidate list sizes 1, 2. Tabu Search performs quite poorly for $v = 3$ (even worse than greedy search on average), and thereafter improves in a monotonic fashion as v is increased to 10. It also appears that the variance in the final solution obtained is very small for $v = 10$. This is certainly odd behaviour. How can it be explained? v is effectively the branching factor at each step, indicating how the neighborhood of each point is sampled before making a move. One possible explanation for the observation is that on average 1 in 3 neighbors of each point in the space are misleading in that they that lead the search into local minima. When only one point is sampled, the search will accept that point unconditionally if it is not in the tabu list. However, when at each step about 3 neighbors are sampled, chances are that picking the lowest of those points leads to a trap (local minimum). As the candidate list size increases beyond 3, more of the neighborhood is being sampled and the better exploration of the neighborhood allows the search to escape such traps by finding relatively rarer lowest points in the neighborhood (perhaps about 1 in 10) that do not lead to local minima.

6.7 Performance of Genetic Algorithms

The influence of population size and selection technique on the performance of Genetic Algorithms was investigated. The three selection techniques discussed in chapter 5 - Proportional Selection, Tournament Selection, and Rank-based Selection were compared for this problem as well with the same parameters. The three sub-plots Figure 6.12 show the performance of each technique for population sizes ranging from 10 to 100 in increments of 5. For each selection technique and population size parameter value 10 independent runs consisting of 100,000 function evaluations were carried out. It is clear that the performance of the Genetic Algorithms is quite robust and independent of the size of the population for this problem. The performance of the proportional and binary tournament

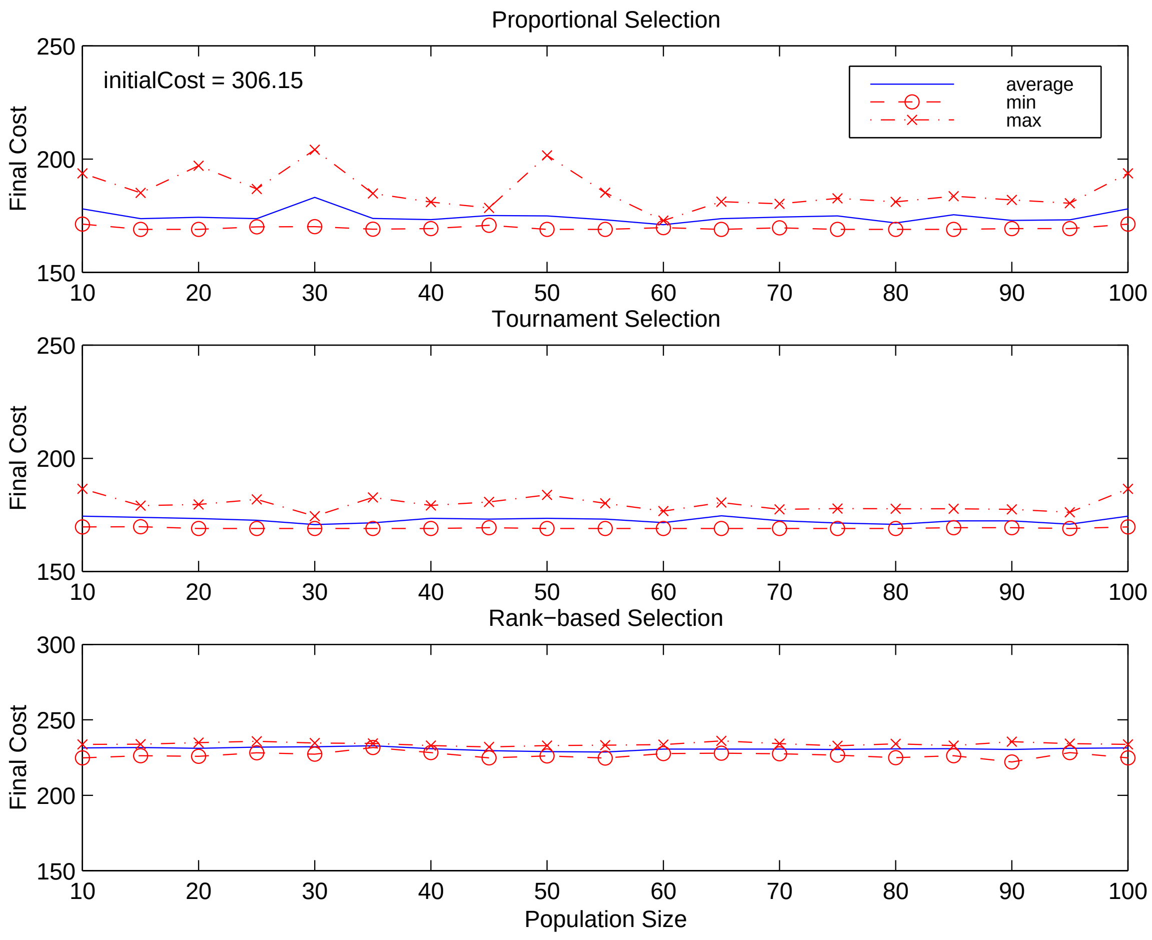


Figure 6.12: Performance of Genetic Algorithms for Fixed Network Topology Optimization

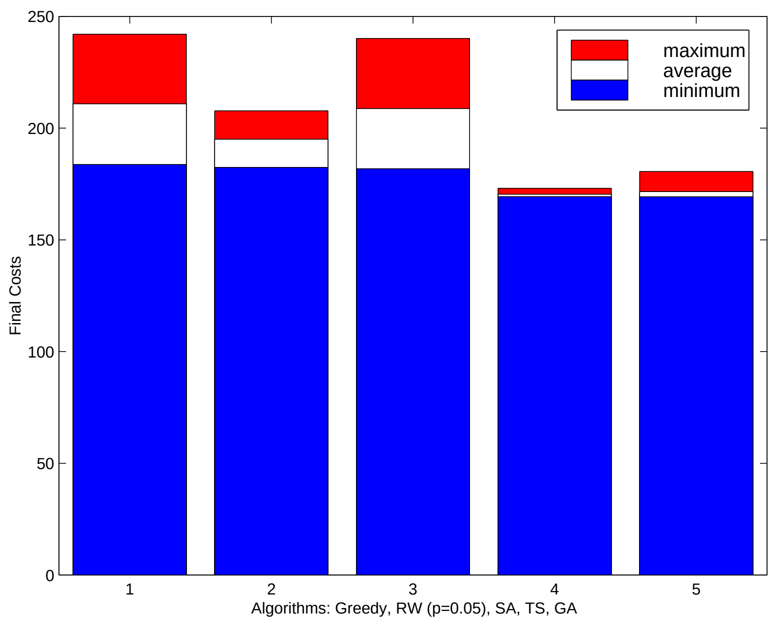


Figure 6.13: Comparison of Search Algorithms for Fixed Network Topology Optimization ($N_{BSC} = 3, N_{BTS} = 50, MAX\_BTS = 20$)

selection schemes is about the same with the latter performing a little better. For reasons mentioned in chapter 5, the rank-based selection technique fares rather poorly for this problem as well.

From an inspection of the final results produced by Genetic Algorithms with Tournament Selection, it appears that they are very likely the globally optimal solutions or very close to it. One support for this speculation is the fact that the minimum cost function solution returned by the GA with tournament selection for each population value is exactly the same.

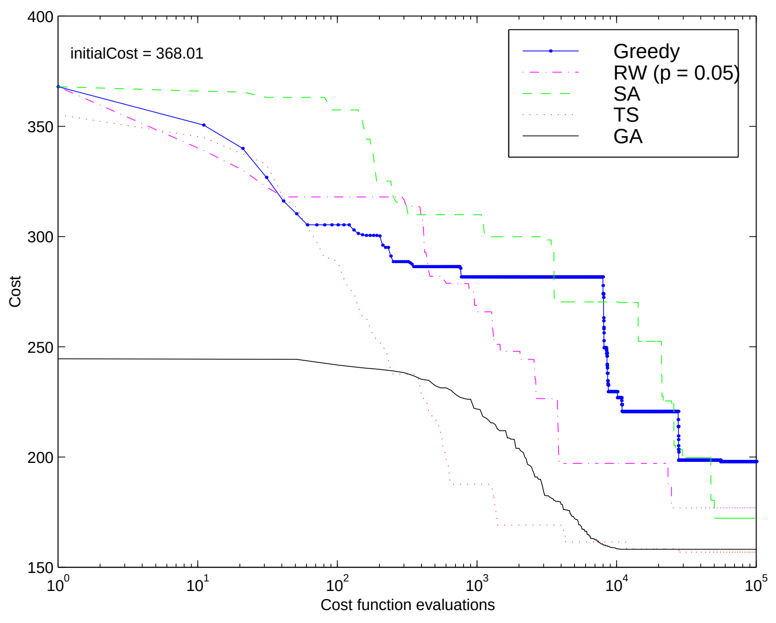


Figure 6.14: Comparison of Search Algorithms for Fixed Network Topology Optimization ($N_{BSC} = 3, N_{BTS} = 50, MAX\_BTS = 20$) (1 Sample Run)

6.8 Comparison of Algorithms

Greedy Search, Random Walk ($p = 0.05$), Simulated Annealing (with geometric cooling schedule, $\alpha_c = 0.99$), Tabu Search (with $K = 4$, $v = 10$), and Genetic Algorithm (with a population size of 50, and elitist binary tournament selection) were compared with each other for their performance on the problem. Figure 6.13 shows the average, minimum and maximum final cost obtained with these algorithms in 100 runs, each consisting of 100,000 cost function evaluations for a network with 3 BSCs, 50 BTSs and MAX_BTS = 20.

It is observed that for this problem, Tabu Search and Genetic Algorithms provide near-optimal final solutions. Random Walk with a small probability of uphill moves performs better than Greedy Search, while the performance of Simulated

Annealing lies somewhere between the two. None of these last three algorithms, however, appear to be able to provide good quality solutions. This is probably due to the search space being really rough and full of local minima. Tabu Search with its aggressive, non-cycling, exploration of the space and Genetic Algorithms with their inherent parallelism both perform better for such a problem. Figure 6.14 shows the best cost seen to date versus number of cost function evaluations for a sample run of each algorithm for this problem. The Genetic Algorithms start off with a low initial cost because one of the 50 points chosen initially at random happened to have a low cost.

To investigate how the performance of these algorithms scales with the size of the problem, these algorithms were compared for two other problem instances: a) smaller search space: $N_{BSC} = 2, N_{BTS} = 20, MAX\_BTS = 15$ b) larger search space: $N_{BSC} = 7, N_{BTS} = 200, MAX\_BTS = 30$. The algorithms were run with the same parameter settings as before for 10 runs of 100,000 cost function evaluations each on both these problem instances. The results are shown in figures 6.15 and 6.16. From figure 6.15, it appears that for the smaller search space while Tabu Search and Genetic Algorithms located the optimal solution in every single run, the other three algorithms were able to do so less frequently (though they did so at least once in the 10 runs). This suggests that perhaps their performance could be improved at least for the smaller problem by using a re-start mechanism (like the multi-start local search described in chapter 5) – this is worth investigating in future work on the subject. The performance comparisons of these algorithms for the larger problem shown in figure 6.16 ranks the algorithms in increasing order of performance as follows: Greedy search, Simulated Annealing, Random Walk (with p = 0.05), Tabu Search, and Genetic Algorithms. Genetic Algorithms were able

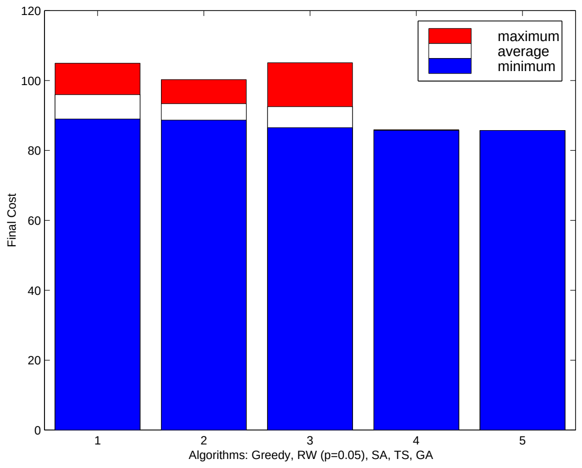


Figure 6.15: Comparison of Search Algorithms for Fixed Network Topology Optimization $(N_{BSC} = 2, N_{BTS} = 20, MAX\_BTS = 15)$

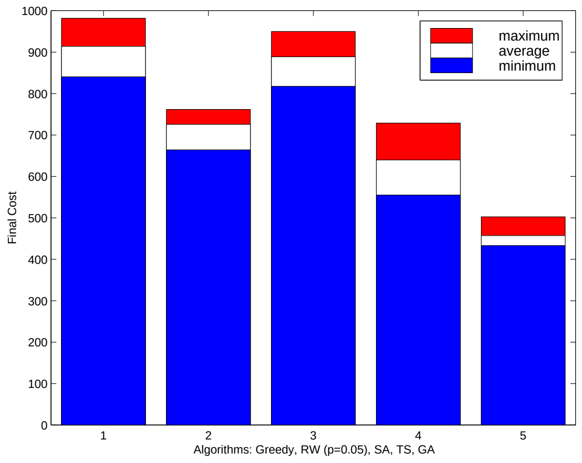


Figure 6.16: Comparison of Search Algorithms for Fixed Network Topology Optimization $(N_{BSC} = 7, N_{BTS} = 200, MAX\_BTS = 30)$

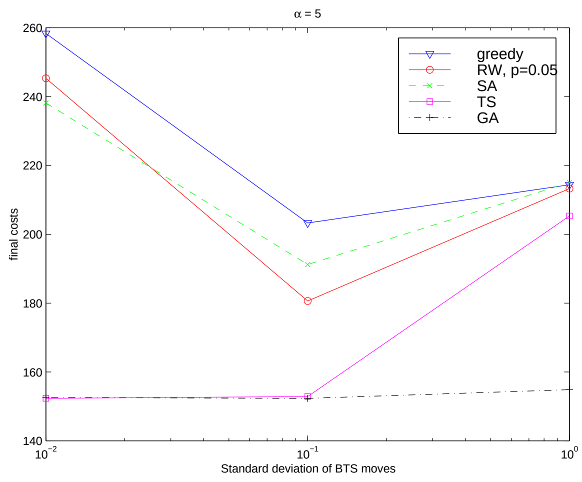


Figure 6.17: Effect of Neighborhood Definition on Search Algorithms for Fixed Network Topology Optimization (Deviation of BSC Moves)

to find near-optimal solutions even for this large problem, while it appeared that Tabu Search would require more iterations to locate these solutions. From these experiments, it is clear that Genetic Algorithms are best-suited for this problem.

6.9 Effect of Neighborhood Definition

As described in section 6.3, the definition of the neighborhood depends upon the values of the parameters σ (deviation of BSC moves), and ω_1, ω_2. The effect of the value of σ upon the performance of these algorithms was examined by running each algorithms 10 times for 100,000 evaluations for $\sigma = [0.01,\ 0.1,\ 1]$. These results are for the default problem size: $N_{BSC} = 2, N_{BTS} = 20, MAX\_BTS = 15$. The average final cost obtained for each is shown in figure 6.17. As σ increases, the neighborhood of each point becomes bigger (as the BSCs are allowed to move further). For the Random Walk Algorithms and Simulated Annealing, a performance improvement is seen from $\sigma = 0.01$ to 0.1 because if the neighborhood is defined as being too small, these algorithms have a greater tendency to be trapped in local optima. Their performance deteriorates as σ is increased further to 1 because the search begins to appear more and more like a random sampling of the space. Tabu Search performs equally well for $\sigma = 0.01$ and 0.1, but starts to perform poorly as σ is increased to 1. Genetic Algorithms, once again, offer the most robust solution as they perform about the same at each value of σ, though it is slightly worse at $\sigma = 1$.

Figures 6.18 - 6.22 show how the performance of each algorithm is affected by varying the values of ω_1 (the probability of making no BTS allocation during a move) and ω_2 (the probability of moving a BTS from one BSC to another BSC). Again, each algorithm was run 10 times for 100,000 function evaluations for the following seven sets of values for (ω_1, ω_2): [(0,0) (0,0.5) (0, 1) (.33, .33) (0.5, 0) (0.5, 0.5) (1, 0)]. The first observation is that for all algorithms (1,0) is a poor choice of BTS assignment neighborhood. This is due to the fact that (1,0) is an extreme choice of parameters in which the base stations assigned to the different base station controllers cannot be changed from the assignment selected in the random initial starting point. The performance of the two best algorithms for this problem, Tabu Search and Genetic Algorithms is also seen to be affected adversely if the values are (0,0). For this pair of values, only exchanges of BTS between BSCs are considered, which means that the number of BTS assigned to each BSC cannot be changed at any time during the search. For Genetic Algorithms the performance is also affected when the values are (0.5, 0), for the same reasons. For all other values of p, performance appears to be quite robust for both TS and GA.

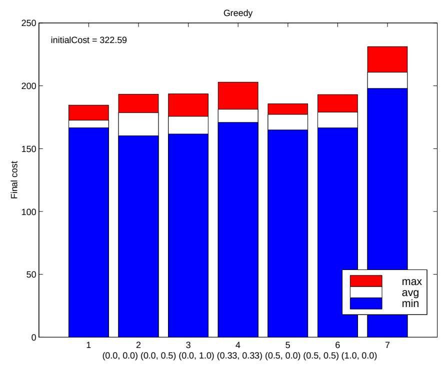


Figure 6.18: Effect of Neighborhood Definition on Greedy Search for Fixed Network Topology Optimization (for different BTS Moves: (ω_1, ω_2))

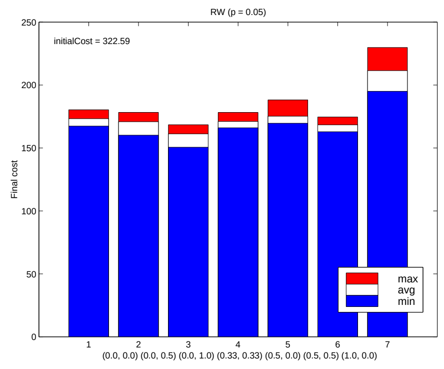


Figure 6.19: Effect of Neighborhood Definition on Random Walk (p = 0.05) for Fixed Network Topology Optimization (for different BTS Moves: (ω_1, ω_2))

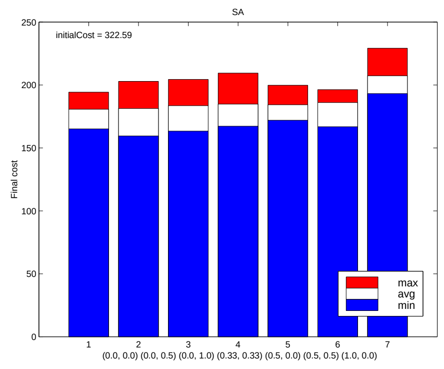


Figure 6.20: Effect of Neighborhood Definition on Simulated Annealing for Fixed Network Topology Optimization (for different BTS Moves: (ω_1, ω_2))

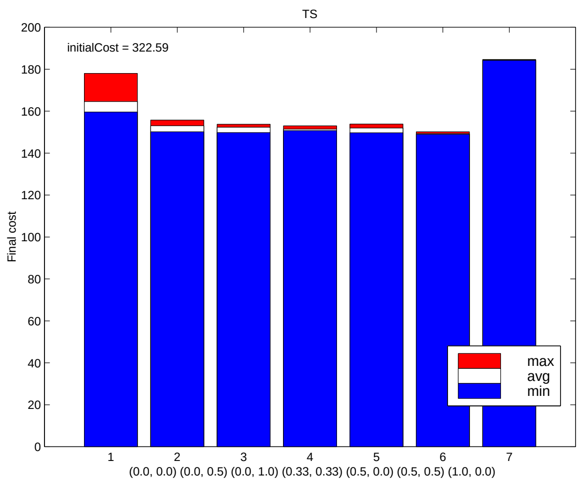


Figure 6.21: Effect of Neighborhood Definition on Tabu Search for Fixed Network Topology Optimization (for different BTS Moves: (ω_1, ω_2))

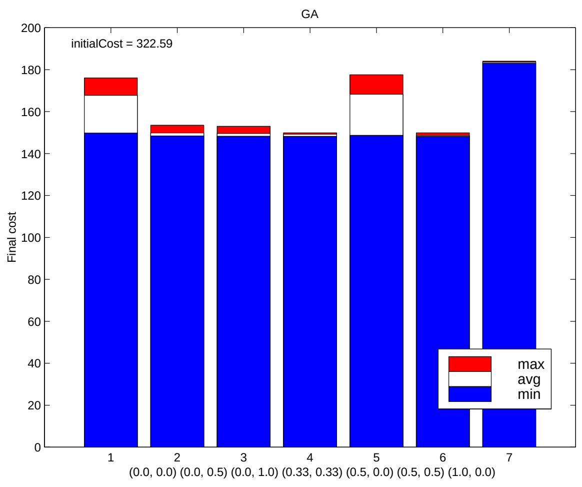


Figure 6.22: Effect of Neighborhood Definition on Genetic Algorithm for Fixed Network Topology Optimization (for different BTS Moves: (ω_1, ω_2))

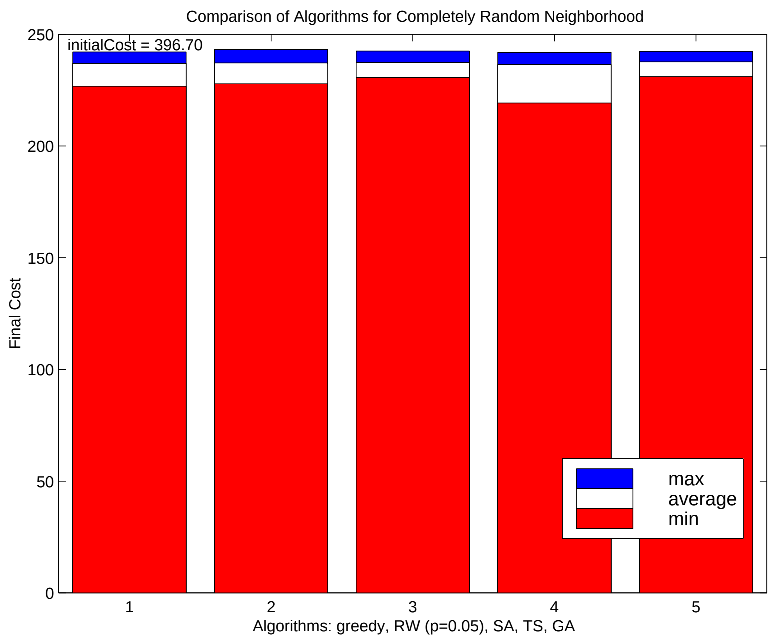
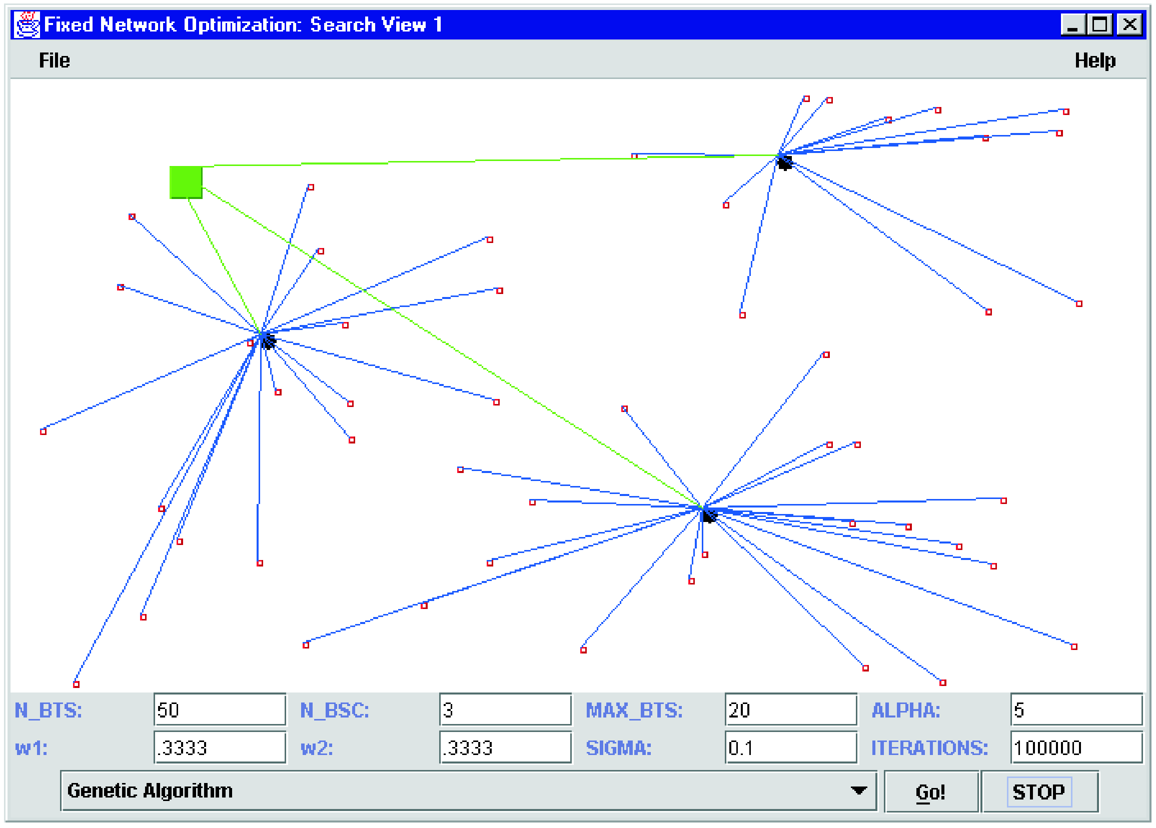


Figure 6.23: Comparison of Search Algorithms for Fixed Network Topology Optimization (Completely Random Neighborhood)

A final experiment to demonstrate the effect of neighborhood definition on the performance of the search algorithms was to compare them for the purely random neighborhood, where each new point is generated independently of the previous point i.e. $N(x) = S$. This is the neighborhood for which all algorithms should perform exactly the same on average. Figure 6.23 which is obtained from 10 runs of 100,000 evaluations each verifies that this is indeed the case.

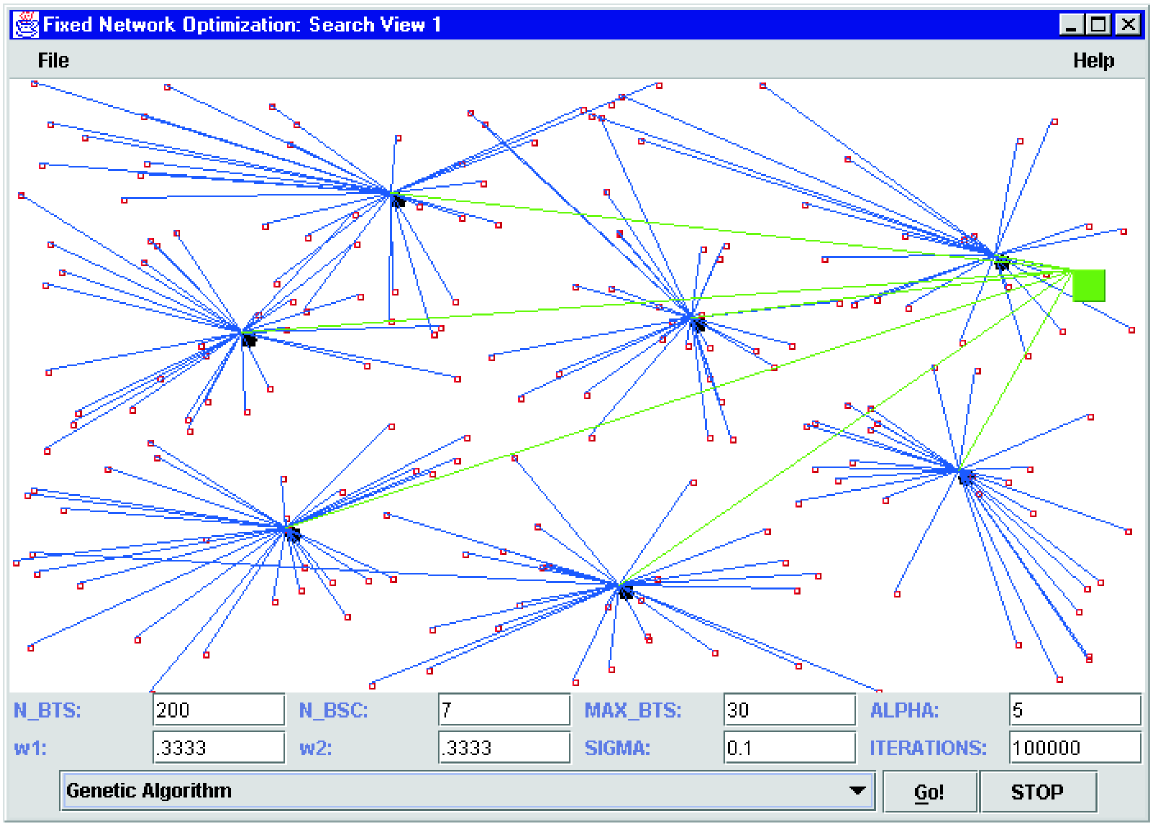
6.10 Final Solutions

Figures 6.24 and 6.25 show sample final solutions obtained by running Genetic Algorithms for 100,000 cost function evaluations. For the smaller problem, it appears that GA has found the optimum solution which consists of BSCs that are spaced apart from each other and in the midst of clusters of base stations that are connected to them. For the larger problem, GA appears to have found a near-optimal solution though it is clear even by visual inspection that the re-assignment of a few base stations to different BSCs would result in a slightly lower over-all cost. From these figures, one could speculate that it may be possible to use a clustering-based heuristic algorithm to provide relatively low-cost solutions in polynomial time. A comparison of the performance of such an algorithm with the solutions obtained through Genetic Algorithms would provide a good indication of the relative strengths of heuristic vs. search techniques for this particular problem.



N_BTS: 50
N_BSC: 3
MAX_BTS: 20
ALPHA: 5
w1: .3333
w2: .3333
SIGMA: 0.1
ITERATIONS: 100000

Figure 6.24: Sample Final Solution obtained with Genetic Algorithms for Fixed Network Topology Optimization ($N_{BSC} = 3, N_{BTS} = 50, MAX\_BTS = 20$)



N_BTS: 200 N_BSC: 7 MAX_BTS: 30 ALPHA: 5
w1: .3333 w2: .3333 SIGMA: 0.1 ITERATIONS: 100000
Genetic Algorithm

Figure 6.25: Sample Final Solution obtained with Genetic Algorithms for Fixed Network Topology Optimization ($N_{BSC} = 7, N_{BTS} = 200, MAX\_BTS = 30$)

Chapter 7

Conclusions

This thesis has investigated four search techniques for global optimization - Random Walk, Simulated Annealing, Tabu Search and Genetic Algorithms. Their application to problems in mobile communications is rapidly growing. As the need for optimal allocation of scarce resources increases with greater demand for better, more comprehensive service in personal communication systems, these techniques may have a significant role to play.

As discussed, the NFL theorems [53] suggest that no claims can be made *a priori* about the strengths of one search algorithm over another for all combinatorial optimization problems. The performance of algorithms varies greatly with the search space and the type of neighborhood scheme utilized. Further, it is impossible to theoretically predict the performance of these algorithms for any but the simplest of search spaces without performing experimental comparisons. It becomes of great importance, therefore, to determine the best algorithm and neighborhood definition for a given problem via experimental analysis. Such an experimental analysis of algorithms was performed for two optimization problems in mobile communications - the optimal location of base stations and minimum cost fixed network topology design.

For difficult optimization problems, there is a tradeoff between the time required for optimization and the optimality of the solutions obtained. Due to computational constraints, search techniques are currently more suitable for problems where some time is available for the optimization process. For situations where optimization needs to be performed in extremely short periods of time, constructive heuristic techniques which offer good (but necessarily near-optimal) solutions much faster may be a better choice. The two problems considered in this thesis are both design problems that are typically encountered in the planning stage of setting up a cellular network. Search techniques are particularly suitable for design problems due to the availability of sufficient time to perform a reasonably efficient search of the space.

The experimental results showed that for the base station location problem a Tabu Search algorithm provided the best overall results, while Genetic Algorithms offered the best performance for the fixed network topology optimization problem. The effect of various algorithm, neighborhood, and problem parameters were also investigated. These experiments yield some insights into the nature of the two problems and the relative strengths and weakness of the algorithms that affect their performance on these search spaces.

There is always room for improvement when it comes to the performance of search techniques. It appears from these experiments that one significant way to improve their performance is to design better neighborhood definitions.

Appendix

The following files of Java source code are included in this appendix:

- BTSLocationOptimization.java : the main class for performing the base station location optimization.
- SearchPoint.java (1) : the class describing the search space and neighborhood definitions for the base station location optimization problem.
- ReadCoverageData.java : used to read raw data describing potential locations for base stations.
- TabuSearch.java (1) : an implementation of Tabu Search for the base station location optimization problem.
- NetworkOptimization.java : the main class for performing the fixed network topology optimization.
- SearchPoint.java (2) : the class describing the search space and neighborhood definitions for the fixed network topology optimization problem.
- TabuSearch.java (2) : an implementation of Tabu Search for the network topology optimization.

- RandomWalk.java : an implementation of Random Walk (common to both optimization problems).
- SimulatedAnnealing.java : an implementation of Simulated Annealing (common to both optimization problems).
- GeneticAlgorithms.java : an implementation of Genetic Algorithms (common to both optimization problems).

```
/** BTSLocationOptimization.java
    Base Station Location Optimization Program
    @author Bhaskar Krishnamachari
    @version 2.0
*/

public class BTSLocationOptimization {

 /** the SearchPoint that starts each run */
 public static SearchPoint xs = new SearchPoint();
 /** stores the values of xs before any changes */
 public static SearchPoint xs_initial = new SearchPoint();
 /** the number of function evaluations to be performed */
 public static int ITER = 10000;
 /** the Neighborhood to be used as defined in SearchPoint */
 private static int NeighborhoodToUse = 1;
 /** the algorithm to run */
 private static String algorithm;

 public static void main(String args[]) {

    System.out.println("%Starting Program....");
    SearchPoint.setupProblem();
    xs = new SearchPoint();

    System.out.println();
    System.out.println("%Initializing first point....");
    xs.init();
    xs_initial.setValueOf(xs);


    algorithm = "Random Walk";
    start();  // start a run
    algorithm = "Simulated Annealing";
    start();
    algorithm = "Tabu Search";
    start();
    algorithm = "Genetic Algorithm";
    start();
  }
```

```
/** initiate a new optimization run */
public static void start() {


   System.out.println("%________________");
   System.out.println("% NEW RUN STARTED ");
   System.out.println("%________________");


   if (algorithm.equals("Random Walk")) {

     double parray[] = {0, 0.03}; // uphill probability
     int repetitions = 10;   // number of times to run
     for (int i = 0; i < parray.length; i++) {
      for (int j = 0; j < repetitions; j++) {
       xs.setValueOf(xs_initial);
       RandomWalk.holds = true;
       RandomWalk rw = new
              RandomWalk(xs, NeighborhoodToUse);
       rw.p = parray[i];
       Thread rwThread = new Thread(rw);
       rwThread.start();
       try { rwThread.join(); }
        catch (InterruptedException e) {
         System.err.println("join() failed!");
        }
      }
    }
   }
   else {
   if (algorithm.equals("Simulated Annealing")) {
     double alpharray[] = {0.5};// cooling parameter
     int repetitions = 10;
     for (int i = 0; i < alpharray.length; i++) {
      for (int j = 0; j < repetitions; j++) {
       xs.setValueOf(xs_initial);
       SimulatedAnnealing.holds = true;
       SimulatedAnnealing sa = new
              SimulatedAnnealing(xs, NeighborhoodToUse);
       sa.alpha = alpharray[i];
       Thread saThread = new Thread(sa);
       saThread.start();
    try { saThread.join(); }
     catch (InterruptedException e) {
      System.err.println("join() failed!");
     }
   }
  }
}
else {
if (algorithm.equals("Genetic Algorithm")) {
  int stypearray[] = {1};   //selection type
  int psizearray[] = {10};  //population size
  int repetitions = 10;
  for (int i = 0; i < stypearray.length; i++) {
   for (int k = 0; k < psizearray.length; k++) {
   for (int j = 0; j < repetitions; j++) {
    xs.setValueOf(xs_initial);
    GeneticAlgorithm.holds = true;
    GeneticAlgorithm ga = new
           GeneticAlgorithm(xs, NeighborhoodToUse);
    ga.selectionType = stypearray[i];
    ga.populationSize = psizearray[k];
    Thread gaThread = new Thread(ga);
    gaThread.start();
    try { gaThread.join(); }
     catch (InterruptedException e) {
      System.err.println("join() failed!");
     }
   }
  }
 }
}
else {
if (algorithm.equals("Tabu Search")) {
  int clsarray[] = {10};  //size of the Candidate List
  int tenurearray[] = {1}; // value of the Tabu Tenure
  int repetitions = 1;
  for (int i = 0; i < clsarray.length; i++) {
   for (int k = 0; k < tenurearray.length; k++) {
   for (int j = 0; j < repetitions; j++) {
    xs.setValueOf(xs_initial);
    TabuSearch.holds = true;
    TabuSearch ts = new
```

```
                TabuSearch(xs, NeighborhoodToUse);
         ts.candidateListSize = clsarray[i];
         ts.tabuTenure = tenurearray[k];
         Thread tsThread = new Thread(ts);
         tsThread.start();
         try { tsThread.join(); }
          catch (InterruptedException e) {
           System.err.println("join() failed!");
          }
        }
       }
      }
     }
     }
     }
     }
  }//start

}
```

```
/** SearchPoint.java (1)
    Describes Each Point in Search Space for the
        BTS Location Optimization Problem.
    @author Bhaskar Krishnamachari
*/

import java.util.*;

public class SearchPoint implements Comparable {

  // CLASS VARIABLES
  // ----------------------------
  /**  all x co-ordinates lie between 0 to XRANGE */
  public static final int X_RANGE = 100;
  /** all y co-ordinates lie between 0 to YRANGE */
  public static final int Y_RANGE = 100;
  /** Total number of possible Base Station locations */
  public static final int N_BTS = 51;
  /** initial number of Base Stations locatied */
  public static final int INITIAL_N_BTS = 26;
  /** exponent of coverPercentage to calculate Cost */
  public static final double ALPHA = 3.0;

  /** the 3-D array containing coverage Information */
  public static int [][][] coverageArray = new
      int [N_BTS][X_RANGE][Y_RANGE];
  /** All possible locations of Base Stations */
  public static boolean [][] locationsArray = new
      boolean [X_RANGE][Y_RANGE];
  /** a Random number generator */
  public static Random randGen = new Random();
  /** mutation probability used to generate neighbor */
  public static double Pm = .01;



  // CLASS METHODS
  // -------------
  /** Create coverageArray and locationsArray from data */
  public static void setupProblem () {
    ReadCoverageData.generateCoverageArray();
    coverageArray = ReadCoverageData.coverageArray;
  locationsArray = ReadCoverageData.getLocationsArray();
}



// INSTANCE VARIABLES
//-------------------
/** describes which locations have been selected currently */
public boolean [] selectedLocations = new boolean [N_BTS];
/** indicates which locations were changed in neighbor() */
public boolean [] changedLocations = new boolean [N_BTS];

/** percentage of area covered by selected BTS */
public double coverPercentage;
/** number of base stations currently selected */
public int currentN_BTS;
/** the cost of the current point in the search space */
public double Cost;

// INSTANCE METHODS
// ----------------
/** initialize a Search - generate a random SearchPoint*/
public void  init () {

  for (int k = 0; k < N_BTS; k++) {
    selectedLocations[k] = false;
    changedLocations[k]  = false;
  }

  int k = randGen.nextInt(N_BTS);
  for (int i=0; i < INITIAL_N_BTS; i++) {
    while (selectedLocations[k] == true )
      k = randGen.nextInt(N_BTS);
    selectedLocations[k] = true; // select random locations
    k = randGen.nextInt(N_BTS);
  }

  currentN_BTS = INITIAL_N_BTS;
  this.evaluateCost(); // calculate the current cost
}

/** generates neighbor of current SearchPoint */
```

```
public SearchPoint neighbor (int NeighborhoodToUse) {

  SearchPoint next = new SearchPoint();

  for (int k=0; k < N_BTS; k++) {
    next.selectedLocations[k] = selectedLocations[k];
    next.changedLocations[k] = false;
  }  //default values

  switch(NeighborhoodToUse) {
    case 0:
    //simple mutations with probability Pm
    // for all bits considered
      for (int k=0; k < N_BTS; k++) {
         if ( (randGen.nextDouble() < Pm)  ) {
            next.selectedLocations[k] =
                 !(selectedLocations[k]);
            next.changedLocations[k] = true;
         }
      }
    break;

    case 1:
    //flip any one (single) bit at random
       int k = randGen.nextInt(N_BTS);
       next.selectedLocations[k] =
            !(selectedLocations[k]);
       next.changedLocations[k] = true;
    break;

    case 2:
    // initiate a new point in a purely random fashion
       next.init();
       for (int k2=0; k2 < N_BTS; k2++) {
          if (selectedLocations[k2] !=
              next.selectedLocations[k2])
               next.changedLocations[k2] = true;
       }
    break;
  }

  next.currentN_BTS = 0;
  for (int k=0; k < N_BTS; k++)
    if (next.selectedLocations[k] == true)
       next.currentN_BTS++;

  next.evaluateCost(); //calculate cost

  return next;
}



/** Evaluate the cost of this assignment */
private void evaluateCost() {
  calculateCoverPercentage();
  Cost = ((double) currentN_BTS)/
      (Math.pow(coverPercentage,ALPHA))*10000.0;
}


/** calculate the percentage of area covered */
private void calculateCoverPercentage() {
  int [][] totalCoverage = new
      int [X_RANGE][Y_RANGE]; //total coverage array
  coverPercentage = 0;

  for (int i=0; i < X_RANGE; i++) {
    for (int j=0; j<X_RANGE; j++)  {
      for (int k=0; k < N_BTS; k++)  {
        if (selectedLocations[k] == true) {
          totalCoverage[i][j] += coverageArray[k][i][j];
        }
      }
      coverPercentage +=
          ( totalCoverage[i][j] > 0) ? 1 : 0;
          //count # of points covered in 100x100 grid
    }
  }
  coverPercentage /= ((double) (X_RANGE*Y_RANGE/100));
}
```

```
  /** to set the values of one search point to another */
  public void setValueOf( SearchPoint sp) {
    for (int k=0; k < N_BTS; k++) {
      this.selectedLocations[k] = sp.selectedLocations[k];
      this.changedLocations[k] = sp.changedLocations[k];
    }
    this.currentN_BTS = sp.currentN_BTS;
    this.Cost = sp.Cost;
    this.coverPercentage = sp.coverPercentage;
  }

  /** for comparing one Search Point to another */
  public int compareTo(Object o)  {
    int returnValue = 0 ;
    SearchPoint s = (SearchPoint) o;
    if (this.Cost < s.Cost) returnValue = -1;
    if (this.Cost == s.Cost) returnValue = 0;
    if (this.Cost > s.Cost) returnValue = 1;

    return returnValue;
  }

}
```

```
/** ReadCoverageData.java
    This program reads the ascii data files that
        describe the possible BS Locations
    @author Bhaskar Krishnamachari
*/

import  java.io.*;

public class ReadCoverageData {

  /** the number of locations */
  public static final int N_BTS = 51;
  /** the x-range of the service area */
  public static final int X_RANGE = 100;
  /** the y-range of the service area */
  public static final int Y_RANGE = 100;
  /** the 3-dimensional array containing all the
   coverage information a 100x100 array for each BTS */
  public static int [][][] coverageArray = new
      int [N_BTS][X_RANGE][Y_RANGE];
  /** the buffered input stream reader to read files */
  private static BufferedReader input = null;

  /** generate the coverageArray from appropriate files */
   public static void generateCoverageArray () {
     int [][] temp  = new int [X_RANGE][Y_RANGE];
     for (int n = 0; n < N_BTS ; n++) {
       String fs = System.getProperty("file.separator");
       String num = (new Integer(n+1)).toString();
       temp = readCoverageFile("coverageData" + fs
           + "cov"+ num + ".txt");
       coverageArray[n] = temp;
     }
   }


   /** get the boolean array of locations from appropriate file */
   public static boolean [][] getLocationsArray() {
     boolean [][] tempBoolean = new boolean [X_RANGE][Y_RANGE];
     int [][] tempInt = new int [X_RANGE][Y_RANGE];
     String fs = System.getProperty("file.separator");
     tempInt = readCoverageFile("coverageData" + fs
          + "randomLocations.txt");
      for (int i=0; i < X_RANGE; i++)
        for (int j=0; j < Y_RANGE; j++)
          tempBoolean[i][j] = (tempInt[i][j] == 1) ? true : false;
      return tempBoolean;
    }


   /** read array from a given file */
    private static int [][] readCoverageFile(String filename){
      int [][] a= new int [X_RANGE][Y_RANGE];
      try { // to catch IO errors
        input = new BufferedReader(new FileReader (filename));
        int i = 0, j = 0;
        char c = (char) input.read();
        while ( c != -1 )    {
          if (Character.isDigit(c)) {
            a[i][j] = ( new Integer( (new
                Character(c)).toString() ) ).intValue();
            if (j != (Y_RANGE-1) )
              j++;  //increment columns correctly
            else {
              i++;
              j = 0;
            }
          }
          if (i == X_RANGE)
            break;
          c = (char) input.read();
        }
      }
      catch (IOException e) {
        System.err.println("IOException encountered");
      }
      return a;
    }

}
```

```
/** TabuSearch.java (1)
    An implementation of Tabu Search for the
        BTS Location Optimization Problem
    @author: Bhaskar Krishnamachari
*/

import java.util.*;

public class TabuSearch implements Runnable {

  // CLASS VARIABLES
  // ---------------
  /** a boolean flag to check in order to stop thread */
  public static boolean holds = true;
  /** store the best cost seen up to each iteration */
  public static double [] xsb ;
  /** size of candidate list */
  public static int candidateListSize;
  /** the tabu tenure */
  public static int tabuTenure;


  // INSTANCE VARIABLES
  // ------------------
  /** The best Search Point seen to date */
  private SearchPoint xs;
  /** The number of iterations to run RW */
  private int iterations;
  /** Neighborhood type to use, as defined in SearchPoint */
  private int NToUse;
  /** the candidate list */
  private SearchPoint [] candidateList;
  /** Tabu list for base stations */
  private int [] btsTabu;

  // CONSTRUCTOR
  // -----------
  public TabuSearch(SearchPoint sp, int NeighborhoodToUse) {
    xs = sp;
    NToUse = NeighborhoodToUse;
  }
```

```
// INSTANCE METHODS
// ----------------
  /** run the Tabu Search */
  public void run() {

    iterations = BTSLocationOptimization.ITER
        / candidateListSize;

    System.out.println("%New Tabu Search run started");

    xsb = new double [iterations];
    initializeTabuLists();
    SearchPoint x = new SearchPoint();
    x.init();
    x.setValueOf(xs);
    SearchPoint xp = new SearchPoint();
    xp.init();
    System.out.println("initialCost" = "+xs.Cost+";");


    for (int i = 0; i < iterations; i++ ) {
      generateCandidateList(x);

      for (int j = 0; j < candidateListSize; j++)
        if ( (isTabu(candidateList[j], i)==false)
             || aspires(candidateList[j]) ) {
          xp.setValueOf(candidateList[j]);
          break;
        }
        if ( (xp.Cost < xs.Cost) ) {
          try{Thread.sleep(1);}
          catch (InterruptedException e) {return;}
          xs.setValueOf(xp);
          xs.changed();
        }

          updateTabuLists(xp,i);
          x.setValueOf(xp);
          xsb[i] = xs.Cost;

      // check if run has been interrupted
    if (holds == false) {
      System.out.println("%iteration,
          current cost: "+i+","+xs.Cost);
      break;
    }

  } // end Tabu Search iteration

  System.out.println();
  System.out.println("Final Cost: "+xs.Cost);
  System.out.println("%________________________");

}


/** initialize the Tabu Lists for the problem */
private void initializeTabuLists() {
  btsTabu = new int [xs.N_BTS];
}

/** generate Candidate list for given Search Point */
private void generateCandidateList(SearchPoint s) {
  candidateList = new SearchPoint [candidateListSize];
  for (int j = 0; j < candidateListSize; j++)  {
    candidateList[j] = new SearchPoint();
    candidateList[j].init();
    candidateList[j].setValueOf(s.neighbor(NToUse));
  }
  Arrays.sort(candidateList);
}

/** check if aspiration criteria are satisfied */
private boolean aspires(SearchPoint s) {
  return (s.Cost < xs.Cost);
}

/** check if a move is Tabu */
private boolean isTabu(SearchPoint s, int iterNumber) {
   for (int j = 0; j < s.N_BTS; j++) {
    if ( (iterNumber - (tabuTenure + btsTabu[j])) < 0 )
         return true;
   }
```

```
     return false;
    }

    /** update the Tabu Lists */
    private void updateTabuLists(SearchPoint sp, int iterNumber) {
      for (int j=0; j< sp.N_BTS; j++) {
        if (sp.changedLocations[j] == true)
           btsTabu[j] = iterNumber;
      }
    }

  }
```

```
/** Fixed Network Topology Minimization Program
    @author Bhaskar Krishnamachari
    Note: Code pertaining to Graphical User Interface
          has been omitted
*/

public class NetworkOptimization {

 /** the SearchPoint that starts each run */
 public static SearchPoint xs = new SearchPoint();
 /** stores the values of xs before any changes */
 private static SearchPoint xs_initial = new SearchPoint();
 /** the number of function evaluations to be performed */
 public static int ITER = 100000;
 /** the Neighborhood to be used as defined in SearchPoint */
 private static int NeighborhoodToUse = 1;
 /** the algorithm to run */
 private static String algorithm ;


 /** the main program to run */
 public static void main(String args[]) {

     // default values for problem parameters
     xs.N_BTS = 50 ;
     xs.N_BSC = 3;
     xs.MAX_BTS = 20;
     xs.ALPHA = 5;
     xs.w1 = .33;
     xs.w2 = .33;
     xs.BSCDeviation = 0.1;

     SearchPoint.setupProblem();
     xs.init();
     xs_initial.setValueOf(xs);

     algorithm = "Random Walk";
     start();
     algorithm = "Simulated Annealing";
     start();
     algorithm = "Tabu Search";
     start();
```

```
   algorithm = "Genetic Algorithm";
   start();
}


/** initiate a new optimization run */
public static void start() {
   System.out.println("%__________________");
   System.out.println("%  NEW RUN STARTED ");
   System.out.println("%__________________");


   xs.setValueOf(xs_initial);

   if (algorithm.equals("Random Walk")) {
     double parray[] = {0, 0.05}; // uphill probability
     int repetitions = 10;   // number of times to run
     for (int i = 0; i < parray.length; i++) {
      for (int j = 0; j < repetitions; j++) {
       xs.setValueOf(xs_initial);
       RandomWalk.holds = true;
       RandomWalk rw = new
           RandomWalk(xs, NeighborhoodToUse);
       rw.p = parray[i];
       Thread rwThread = new Thread(rw);
       rwThread.start();
       try { rwThread.join(); }
       catch (InterruptedException e) {
        System.err.println("join() failed!");
       }
      }
    }
   }
   else {
   if (algorithm.equals("Simulated Annealing")) {
     double alpharray[] = {0.99};// cooling parameter
     int repetitions = 10;
     for (int i = 0; i < alpharray.length; i++) {
      for (int j = 0; j < repetitions; j++) {
       xs.setValueOf(xs_initial);
       SimulatedAnnealing.holds = true;
       SimulatedAnnealing sa =
        new SimulatedAnnealing(xs, NeighborhoodToUse);
    sa.scheduleType = 0;
    sa.alpha = alpharray[i];
    Thread saThread = new Thread(sa);
    saThread.start();
    try { saThread.join(); }
    catch (InterruptedException e) {
      System.err.println("join() failed!");
    }
   }
  }
}
else {
if (algorithm.equals("Genetic Algorithm")) {
  int stypearray[] = {1};   //selection type
  int psizearray[] = {30};  //population size
  int repetitions = 10;
  for (int i = 0; i < stypearray.length; i++) {
   for (int k = 0; k < psizearray.length; k++) {
   for (int j = 0; j < repetitions; j++) {
    xs.setValueOf(xs_initial);
    GeneticAlgorithm.holds = true;
    GeneticAlgorithm ga = new
        GeneticAlgorithm(xs, NeighborhoodToUse);
    ga.selectionType = stypearray[i];
    ga.populationSize = psizearray[k];
    Thread gaThread = new Thread(ga);
    gaThread.start();
    try { gaThread.join(); }
    catch (InterruptedException e) {
         System.err.println("join() failed!");
    }
   }
  }
 }
}
else {
if (algorithm.equals("Tabu Search")) {
  int clsarray[] = {10};  //size of the Candidate List
  int tenurearray[] = {4}; // value of the Tabu Tenure
  int repetitions = 1;
  for (int i = 0; i < clsarray.length; i++) {
        for (int k = 0; k < tenurearray.length; k++) {
        for (int j = 0; j < repetitions; j++) {
         xs.setValueOf(xs_initial);
         TabuSearch.holds = true;
         TabuSearch ts = new
             TabuSearch(xs, NeighborhoodToUse);
         ts.candidateListSize = clsarray[i];
         ts.tabuTenure = tenurearray[k];
         Thread tsThread = new Thread(ts);
         tsThread.start();
         try { tsThread.join(); }
         catch (InterruptedException e) {
          System.err.println("join() failed!");
         }
        }
       }
      }
     }
     }
     }
     }
  }

}
```

```
/** SearchPoint.java (2)
    Describes Each Point in Search Space for the
        Fixed Network Topology Optimization Problem.
    @author Bhaskar Krishnamachari
    Note: the code pertaining to the Graphical User
        Interface is omitted
*/


import java.awt.geom.*;
import java.util.*;

public class SearchPoint implements Comparable {

  // CLASS VARIABLES
  // ----------------------------
  /**  all x co-ordinates lie between 0 to XRANGE */
  public static final double X_RANGE = 10F;
  /** all y co-ordinates lie between 0 to YRANGE */
  public static final double Y_RANGE = 10F;
  /** number of BTS chosen */
  public static int N_BTS;
  /** maximum number of BTS per BSC */
  public static int MAX_BTS;
  /** number of BSC's chosen */
  public static int N_BSC;
  /** ALPHA is the relative cost of links between
      MSCs & BSCs and between BSCs & BTSs  */
  public static double ALPHA;
  /** for locations of BTS */
  public static Point2D.Double BTSLocation[];
  /** for location of the MSC */
  public static Point2D.Double MSCLocation;
  /** a Random number generator */
  public static Random randGen = new Random();
  /** variance for locations of BSC, default 1 */
  public static double BSCDeviation;
  /** weights for changing BTS assignments */
  public static double w1, w2;
```

```
// CLASS METHODS
// -------------
/** Setup the fixed locations of MSC and BTS */
public static void setupProblem () {
  MSCLocation = new Point2D.Double(
      randGen.nextDouble()*10 , randGen.nextDouble()*10);
  BTSLocation = new Point2D.Double [N_BTS];
  for (int i=0; i < N_BTS; i ++)
    BTSLocation[i] = new Point2D.Double(
        randGen.nextDouble()*10, randGen.nextDouble()*10);
}



// INSTANCE VARIABLES
//-------------------
/** for locations of BSC */
public Point2D.Double BSCLocation[];
/** current assignment of BTS to the BSCs */
public int assignedBTS[][];
/** cost of links from BTS to BSCs*/
public double partialCost1[];
/** cost of links from BSC to MSCs for BSC1 only*/
public double partialCost2[];
/** the Cost function evaluation for this SearchPoint */
public double Cost;


// INSTANCE METHODS
// ----------------
/** initialize a Search - generate a random SearchPoint*/
public void  init () {
  //get the values for the instance variables
  BSCLocation = new Point2D.Double [N_BSC];
  assignedBTS = new int[N_BSC][];
  partialCost1 = new double [N_BSC];
  partialCost2 = new double [N_BSC];


  // assign BTS
  int count[]  = new int [N_BSC];
  for (int i = 0; i < N_BSC; i++)
    count[i] = 0;
  int allocation[] = new int [N_BTS];
  for (int j = 0; j < N_BTS; j++) {
    int t = randGen.nextInt(N_BSC);
    while ( count[t] == MAX_BTS ) {
      t = randGen.nextInt(N_BSC);
    }
    allocation[j] = t;
    count[t]++;
  }


  for (int i = 0; i < N_BSC; i++)
    assignedBTS[i] = new int [count[i]];

  for (int i = 0; i < N_BSC; i++)
    count[i] = 0;
  for (int j = 0; j < N_BTS; j++)
    assignedBTS[allocation[j]][count[allocation[j]]++] = j;


  // locate BSC and calculate costs
  Cost = 0F;
  for (int i = 0; i < N_BSC; i++) {
    BSCLocation[i] = new Point2D.Double(randGen.nextDouble()
        *X_RANGE, randGen.nextDouble()*Y_RANGE);
    partialCost1[i] = 0;
    for (int j = 0; j < assignedBTS[i].length; j++)
      partialCost1[i] += (double) BSCLocation[i].
          distance(BTSLocation[assignedBTS[i][j]]);
    partialCost2[i] = (double) BSCLocation[i].
        distance(MSCLocation);
    Cost += partialCost1[i] + ALPHA*partialCost2[i];
  }
}


/** generates neighbor of current SearchPoint */
public SearchPoint neighbor (int NeighborhoodToUse) {

  SearchPoint next = new SearchPoint();
```

```
if (NeighborhoodToUse == 0) {
  next.init();
  return next;
}

next.BSCLocation = this.BSCLocation;
next.partialCost1 = new double [N_BSC];
next.partialCost2 = new double [N_BSC];
next.assignedBTS =  new int[N_BSC][];

int BSC1 = randGen.nextInt(N_BSC);

//generate move of BSC
double newX = (double) (randGen.nextGaussian()
    *BSCDeviation + BSCLocation[BSC1].getX());
double newY = (double) (randGen.nextGaussian()
    *BSCDeviation + BSCLocation[BSC1].getY());

// check for limits
newX = (newX < 0) ? 0 : newX;
newX = (newX > X_RANGE) ? X_RANGE : newX;
newY = (newY < 0) ?  0 : newY;
newY = (newY > Y_RANGE) ? Y_RANGE : newY;


// move!
next.BSCLocation[BSC1] = new
    Point2D.Double(newX, newY);


// pick another BSC
int BSC2 = randGen.nextInt(N_BSC);
for ( ; BSC1 == BSC2; BSC2 = randGen.nextInt(N_BSC) )
  ;

double moveType = randGen.nextDouble();
int r1 = randGen.nextInt(assignedBTS[BSC1].length);
int r2 =  randGen.nextInt(assignedBTS[BSC2].length);

// randomly pick one BTS from both BSC1 and BSC2 if
//  it can be assigned to the other
int ch1 = (assignedBTS[BSC2].length < MAX_BTS) ?
```

```
     assignedBTS[BSC1][r1] : -1;  //-1 implies null BTS
int ch2 = (assignedBTS[BSC1].length < MAX_BTS) ?
     assignedBTS[BSC2][r2] : -1;
 // normally ch1 ranges from 0 to N_BTS-1

//Deal with the three cases - no change, move one, swap
//    assignments of BTS to BSCs
// 2: move from BSC1 to BSC2
if ((moveType > w1) && (moveType <= w1+w2) && (ch1 !=-1)) {
  int lastElement= assignedBTS[BSC1]
      [assignedBTS[BSC1].length - 1];
  for (int i=0, j=0; i < N_BSC; i++) {
    if ( (i == BSC1)  ) {
      next.assignedBTS[i]  = new int [
          assignedBTS[BSC1].length - 1];
      for (j = 0; j < (next.assignedBTS[i].length); j++){
        next.assignedBTS[i][j] = assignedBTS[i][j];
      }
      if (r1 != j) {
        next.assignedBTS[i][r1] = lastElement;
      }
     }
     else {
       if ( (i == BSC2) ) {
        next.assignedBTS[i] = new int
            [assignedBTS[BSC2].length+1];
        for (j = 0; j < (next.assignedBTS[i].length-1); j++){
          next.assignedBTS[i][j] = assignedBTS[i][j];
        }
        next.assignedBTS[i][assignedBTS[i].length] = ch1;
      }
      else { // i!=BSC1 or BSC2
        next.assignedBTS[i]  = new int
            [assignedBTS[i].length];
        for (j = 0; j < (next.assignedBTS[i].length); j++) {
          next.assignedBTS[i][j] = assignedBTS[i][j];
        }
      }
    }
  }
}
else  {
      // 1: do nothing
      for (int i=0; i < N_BSC; i++) {
        next.assignedBTS[i] = new int
            [assignedBTS[i].length];
        for (int j = 0; j < (next.assignedBTS[i].length); j++)
          next.assignedBTS[i][j] = assignedBTS[i][j];
      }
      // 3: exchange BTSs between two BSCs
      if ((moveType > w1+w2) && (ch1 != -1) && (ch2 != -1)){
          next.assignedBTS[BSC1][r1] = ch2;
          next.assignedBTS[BSC2][r2] = ch1;
      }
    }


    // calculating cost of neighbor
    next.Cost = 0F;

    for (int i = 0; i < N_BSC; i++) {
      next.partialCost1[i] = 0;
      for (int j = 0; j < next.assignedBTS[i].length; j++)
        next.partialCost1[i] += (double) next.BSCLocation[i].
            distance(next.BTSLocation[next.assignedBTS[i][j]]);
      next.partialCost2[i] = (double) next.BSCLocation[i].
          distance(next.MSCLocation);
      next.Cost += next.partialCost1[i] +
          ALPHA*next.partialCost2[i];
    }

    return next;
  }

/** to set the values of one search point to another */
  public void setValueOf( SearchPoint sp) {
    this.BSCLocation = new Point2D.Double [N_BSC];
    this.assignedBTS = new int[N_BSC][];
    this.partialCost1 = new double [N_BSC];
    this.partialCost2 = new double [N_BSC];

    this.Cost = sp.Cost;
    for (int i = 0; i < N_BSC; i++) {
      this.partialCost1[i] = sp.partialCost1[i];
      this.partialCost2[i] = sp.partialCost2[i];
      this.BSCLocation[i] = sp.BSCLocation[i];
      this.assignedBTS[i] = new int
          [sp.assignedBTS[i].length];
      for (int j = 0; j < this.assignedBTS[i].length; j++) {
        this.assignedBTS[i][j] = sp.assignedBTS[i][j];
      }
    }
  }


  /** for comparisons */
  public int compareTo(Object o)  {
    int returnValue = 0 ;
    SearchPoint s = (SearchPoint) o;
    if (this.Cost < s.Cost) returnValue = -1;
    if (this.Cost == s.Cost) returnValue = 0;
    if (this.Cost > s.Cost) returnValue = 1;

    return returnValue;
  }

}
```

```
/** TabuSearch.java (2)
    An implementation of Tabu Search for the
        Network Topology Optimization Problem
    @author: Bhaskar Krishnamachari
*/

import java.awt.geom.*;
import java.util.*;

public class TabuSearch implements Runnable {

  // CLASS VARIABLES
  // ---------------
  /** a boolean flag to check in order to stop the thread */
  public static boolean holds = true;
  /** store the best cost seen up to each iteration */
  public static double [] xsb ;
  /** size of candidate list */
  public static int candidateListSize;
  /** the tabu tenure */
  public static int tabuTenure;

  // INSTANCE VARIABLES
  // ------------------
  /** The best Search Point seen to date */
  private SearchPoint xs;
  /** The number of iterations to run RW */
  private int iterations;
  /** Neighborhood type to use - as defined in SearchPoint*/
  private int NToUse;
  /** the candidate list */
  private SearchPoint [] candidateList;
  /** Tabu list for BTS */
  private int [] btsTabu;
  /** Tabu list for BSC locations */
  private LinkedList locationTabu = new LinkedList();


  // CONSTRUCTOR
  // -----------
  /** constructor */
  public TabuSearch(SearchPoint sp, int NeighborhoodToUse) {
```

```
  xs = sp;
  NToUse = NeighborhoodToUse;
}


// INSTANCE METHODS
// ----------------
/** run the Tabu Search */
public void run() {
  System.out.println("%New Tabu Search run started");

  iterations = NetworkOptimization.ITER/candidateListSize;
  xsb = new double [iterations];
  initializeTabuLists();
  SearchPoint x = new SearchPoint();
  x.init();
  x.setValueOf(xs);
  SearchPoint xp = new SearchPoint();
  xp.init();
  System.out.println("%Initial Cost: "+xs.Cost);

  for (int i = 0; i < iterations; i++ ) {
    generateCandidateList(x);

    for (int j = 0; j < candidateListSize; j++)
      if ( (isTabu(candidateList[j], i)==false)
          || aspires(candidateList[j]) ) {
        xp.setValueOf(candidateList[j]);
        break;
      }

      if ( (xp.Cost < xs.Cost) ) {
        try{Thread.sleep(1);}
        catch (InterruptedException e) {return;}
        xs.setValueOf(xp);
        xs.changed();
      }

        updateTabuLists(x, xp,i);
        x.setValueOf(xp);
        xsb[i] = xs.Cost;
```

```
   // check if run has been interrupted
    if (holds == false) {
      System.out.println("%iteration,
          current cost: "+i+","+xs.Cost);
      break;
    }

  } // end Tabu Search iteration

  System.out.println();
  System.out.println("Final Cost: "+xs.Cost);
  System.out.println("%________________");

}


/** initialize the Tabu Lists for the problem */
private void initializeTabuLists() {
  locationTabu = new LinkedList();
  btsTabu = new int [xs.N_BSC][xs.N_BTS];
  for (int k = 0; k < tabuTenure; k++)
    locationTabu.add(null);
}


/** generate Candidate list for given Search Point */
private void generateCandidateList(SearchPoint s) {

  candidateList = new SearchPoint [candidateListSize];
  for (int j = 0; j < candidateListSize; j++)  {
    candidateList[j] = new SearchPoint();
    candidateList[j].init();
  }

  for ( int j = 0; j < candidateListSize; j++ )   {
    candidateList[j].init();
    candidateList[j].setValueOf(s.neighbor(NToUse));
  }
  Arrays.sort(candidateList);
}

/** check if aspiration criteria are satisfied */
```

```
private boolean aspires(SearchPoint s) {
  return (s.Cost < xs.Cost);
}

/** check if a move is Tabu */
private boolean isTabu (SearchPoint s, int iterNumber) {
 double tabuDistance = s.BSCDeviation*.1;
 Point2D.Double [] myBSCLocation  = new
     Point2D.Double [s.N_BSC];

 if (locationTabu.getLast() != null) {
   for (int i = 0; i < s.N_BSC; i++) {
     myBSCLocation[i]  = ( (Point2D.Double [])
         locationTabu.getLast())[i];
     if ( myBSCLocation[i].
         distance(s.BSCLocation[i])  < tabuDistance )
       return true;
   }
 }

 for (int i = 0; i < s.N_BSC; i++) {
   for (int j = 0; j < s.assignedBTS[i].length; j++) {
    if ( (iterNumber - (tabuTenure + btsTabu[i]
        [s.assignedBTS[i][j]])) < 0 )
         return true;
   }
 }
 return false;
}

/** update the Tabu Lists */
private void updateTabuLists(SearchPoint s,
    SearchPoint sp, int iterNumber) {
 // tenure update for location linkedlist
 Point2D.Double [] bscLocations = new
     Point2D.Double [sp.N_BSC];
 for (int i = 0; i < sp.N_BSC; i++)
   bscLocations[i] = sp.BSCLocation[i];

 int formerTabuTenure = locationTabu.size();
 int changeInTenure = Math.abs(
     formerTabuTenure - tabuTenure);
    if (tabuTenure > formerTabuTenure)
      for (int i = 0; i < changeInTenure; i++)
        locationTabu.addFirst(null);
    else
      for (int i = 0; i < changeInTenure; i++)
        locationTabu.removeFirst();

    // update the location linkedlist
    locationTabu.removeFirst();
    locationTabu.addLast(bscLocations);

    // update the btsTabu
    HashSet HSp = new HashSet();
    for (int i = 0; i < s.N_BSC; i++) {
      for (int j=0; j < sp.assignedBTS[i].length; j++)
        HSp.add(new Integer(sp.assignedBTS[i][j]));
      for (int j = 0; j < s.assignedBTS[i].length; j++) {
         if (HSp.contains(new
             Integer(s.assignedBTS[i][j])) == false)
           btsTabu[i][s.assignedBTS[i][j]] = iterNumber;
      }
     }
  }
}
```

```
/** An implementation of Random Walk
       ( common for both optimizations )
    @author: Bhaskar Krishnamachari
*/

import java.util.*;


public class RandomWalk implements Runnable{

  // CLASS VARIABLES
  // ---------------
  /** probability of uphill moves */
  public double p;
  /** a boolean flag to check in order to stop thread */
  public static boolean holds = true;
  /** store the best cost seen up to each iteration */
  public static double xsb[] ;

  // INSTANCE VARIABLES
  // ------------------
  /** The best Search Point seen to date */
  private SearchPoint xs;
  /** The number of iterations to run RW */
  private int iterations;
  /** Neighborhood type to use as defined in SearchPoint*/
  private int NToUse;

  // CONSTRUCTOR
  //-------------
  /** constructor */
  public RandomWalk(SearchPoint sp, int NeighborhoodToUse){
    xs = sp;
    NToUse = NeighborhoodToUse;
  }

  // INSTANCE METHODS
  // ------------------
  /** run the Random Walk Algorithm */
  public void run() {
    System.out.println("%New Random Walk run started");
    iterations = NetworkOptimization.ITER;
       // NOTE: this would be BTSLocationOptimization.ITER
       // for Base Station Location Optimization


    SearchPoint x = new SearchPoint();
    x.setValueOf(xs);
    xsb = new double [iterations];
    SearchPoint xp = new SearchPoint();
    System.out.println("%Initial Cost: "+ xs.Cost);

    for (int i = 0; i < iterations; i++) {
      xp.setValueOf(x.neighbor(NToUse));
      if (xp.Cost <= x.Cost) {
        x.setValueOf(xp);
      }

      if (xp.Cost > x.Cost) {
        if (SearchPoint.randGen.nextDouble() < p)
          x.setValueOf(xp);
      }

      if ( (xp.Cost < xs.Cost) ) {
        try{Thread.sleep(1);}
        catch (InterruptedException e) {return;}
        xs.setValueOf(xp);
        xs.changed();
      }
      xsb[i] = xs.Cost;
      if (holds == false) {
        System.out.println("% iteration,
             current cost: "+i+","+ xs.Cost);
        break;
      }
    }
    System.out.println();
    System.out.println("%Final cost: " + xs.Cost);
    System.out.println("%_____________________");
  }

}
```

```
/** An implementation of Simulated Annealing
      ( common to both optimizations )
    @author: Bhaskar Krishnamachari
*/

import java.util.*;

public class SimulatedAnnealing implements Runnable {

  // CLASS VARIABLES
  // ---------------
  /** a boolean flag to check in order to stop thread */
  public static boolean holds = true;
  /** store the best cost seen up to each iteration */
  public static double xsb[] ;


  // INSTANCE VARIABLES
  // ------------------
  /** the best Search Point seen to date */
  private SearchPoint xs, sp;
  /** the number of iterations to run SA */
  private int iterations;
  /** neighborhood type to use - as defined in SearchPoint*/
  private int NToUse;
  /** probability of uphill moves,
      determined by Metropolis criterion */
  private double p;
  /** type of cooling schedule to be used */
  public int scheduleType;
  /** the length of Markov Chains */
  private int MarkovLength = 100;
  /** The standard deviation at infinite temperature */
  private double sigmaInf;
  /** standard deviation of costs at a given temperature */
  private double sigmaVal = 0;
  /** initial temperature */
  private double To = 0;
  /** current temperature */
  private double Temperature = 0;
  /** cooling distant factor */
  public double delta = 0.1;
```

```
/** cooling rate parameter */
public double alpha = 0.99;

// CONSTRUCTOR
// -----------
/** constructor*/
public SimulatedAnnealing(SearchPoint sp,
    int NeighborhoodToUse) {
  xs = sp;
  NToUse = NeighborhoodToUse;
}


// INSTANCE METHODS
// ----------------
/** run the simulated annealing algorithm */
public void run() {

  System.out.println("%New Simulated Annealing
      run started");

  iterations = NetworkOptimization.ITER;
     // NOTE: this would be BTSLocationOptimization.ITER
     // for Base Station Location Optimization
  MarkovLength = iterations/1000;

  // Preparations to start on outer loop
  //    identify sigmaInf, To, initialize x,
  SearchPoint x = new SearchPoint();
  xsb = new double [iterations];
  double costsArray [] = new double [MarkovLength];
  for (int i = 0; i < costsArray.length ; i++) {
    x.init();
    costsArray[i] = x.Cost;
  }

  sigmaInf = stdDev(costsArray);
  To = 10*sigmaInf; // definition of initial Temperature
  x.setValueOf(xs);
  SearchPoint xp = new SearchPoint();
  System.out.println("%Initial Cost: "+xs.Cost);
```

```
int i = 0;
for ( Temperature= (double) To, sigmaVal = sigmaInf;
    i < iterations; Temperature = cool(Temperature)) {

  if ( (new Double(Temperature)).isNaN() )
    Temperature = 0.0;

  costsArray = new double [MarkovLength];
  for (int j = 0; j < costsArray.length ; j++, i++) {
    xp.setValueOf(x.neighbor(NToUse));

    if (xp.Cost <= x.Cost) {
      x.setValueOf(xp);
    }

    if (xp.Cost > x.Cost) {
      p = Math.min(1F, (double) Math.exp(
          - (xp.Cost - x.Cost) / Temperature) ) ;
      if (SearchPoint.randGen.nextDouble() < p)
        x.setValueOf(xp);
    }

    if ( (xp.Cost < xs.Cost) ) {
      try{Thread.sleep(1);}
      catch (InterruptedException e) {return;}
      xs.setValueOf(xp);
      xs.changed();
    }

    costsArray[j] = x.Cost;


    xsb[i] = xs.Cost;
  }

  // update sigma
  sigmaVal = stdDev(costsArray);

  // check if run has been interrupted
  if (holds == false) {
    System.out.println("%iteration,
        current cost: "+i+","+xs.Cost);
        break;
      }
    } // end annealing
    System.out.println();
    System.out.println("%Final cost: "+xs.Cost);
    System.out.println("%____________________");
  }

  /** return standardDeviation of array */
  private static double stdDev ( double costArray[] ) {
    double sigma = 0, mu =0, cov = 0;
    int size = costArray.length;
    for (int i=0; i < size; i++) {
      mu += costArray[i];
      cov += (double) Math.pow((double) costArray[i],
          (double) 2);
    }
      cov = cov / size;  // normalize to get average
      mu  =  mu / size;
    sigma = (double) Math.pow( (double) cov - Math.pow(
        (double) mu, (double) 2 ) , (double) 0.5);

    return sigma;
  }

  /** return temperature decrement according to type
        of Cooling Schedule*/
  private double cool (double T) {
    switch (scheduleType) {
      case 0:  return (double) (Temperature*alpha);
      case 1:  return (double)
          ( T/(1 + T*Math.log(1+delta)/(3*sigmaVal)) );
      case 2:  return (double)
          (T*Math.exp(-delta*T/sigmaVal));
    }
      //default return
    return (double)( T/(1 + T*Math.log(1+delta)/(3*sigmaVal)));
  }

}
```

```
/** An implementation of Genetic Algorithms
       ( common to both optimizations )
    @author: Bhaskar Krishnamachari
    Note: implementation of selection schemes is based on
      descriptions in the Handbook of Evolutionary Computation.
*/

import java.util.*;

public class GeneticAlgorithm implements Runnable {

  // CLASS VARIABLES
  // ---------------
  /** a boolean flag to check in order to stop thread */
  public static boolean holds = true;
  /** store the best cost seen up to each generation */
  public static double xsb[] ;
  /** the size of the population */
  public static int populationSize;
  /** type of selection used: Proportional,
      Tournament, or Rank-based  */
  public static int selectionType;
  /** exponent by which to scale cost function */
  public static double scalingExponent = 5;
  /** size parameter for Tournament Selection */
  public static int tournamentSize = 2;
  /** Expected number of offspring of best
       chromosome in Rank-based Selection */
  public static double betaRank = 1.5;
  /** Defines whether the Genetic Algorithm employs
      the elitist approach */
  public static boolean elitist = true;

  // INSTANCE VARIABLES
  // ------------------
  /** The best Search Point seen to date */
  private SearchPoint xs;
  /** The number of generations to run the GA */
  private int Generations;
  /** Neighborhood type to use - as defined in SearchPoint*/
  private int NToUse;
```

```
// CONSTRUCTOR
// -----------
/** constructor */
public GeneticAlgorithm(SearchPoint sp, int NeighborhoodToUse) {
  xs = sp;
  NToUse = NeighborhoodToUse;
}


// INSTANCE METHODS
// ----------------
/** run the Genetic Algorithm */
public void run() {
  System.out.println("%New Genetic Algorithm run started");

  Generations = (int)
      (NetworkOptimization.ITER / populationSize);
     // NOTE: this would be BTSLocationOptimization.ITER
     // for Base Station Location Optimization
  xsb = new double [Generations];
  System.out.println("%Initial Cost: " + xs.Cost );

  // Define population for the run
  SearchPoint population[] = new
      SearchPoint [populationSize];
  for (int i = 0; i < populationSize; i++)  {
    population[i] = new SearchPoint();
    population[i].init();
  }
  SearchPoint [] nextPopulation  = new
      SearchPoint [populationSize];
  for (int i = 0; i < populationSize; i++)  {
    nextPopulation[i] = new SearchPoint();
    nextPopulation[i].init();
  }
  population[0].setValueOf(xs);
  //sort the initial population into ascending order of cost
  Arrays.sort(population) ;
  // start genetic algorithm:
  for (int t = 0; t < Generations; t++) {
    // select parents
    nextPopulation = selectNew(population);
    // generate offspring
    nextPopulation = mutate(nextPopulation);

    // retain best chromosome if elitist strategy is used
    if (elitist)
      nextPopulation[populationSize-1].
          setValueOf(population[0]);

    //establish new generation
    for (int i = 1; i < populationSize; i++)
      population[i].setValueOf(nextPopulation[i]);
    Arrays.sort(population);

    if ( (population[0].Cost < xs.Cost) ) {
        xs.setValueOf(population[0]);
        xs.changed();
    }


      xsb[t] = xs.Cost;

    // check if run has been interrupted
    if (holds == false) {
      System.out.println("%generation,
          current cost: "+t+","+xs.Cost);
      break;
    }
  } // end evolution

  System.out.println();
  System.out.println("%Final cost: "+xs.Cost);
  System.out.println("%____________________");
}

/** select parents in generation based on selectionType*/
private SearchPoint[] selectNew (SearchPoint [] P) {
  switch (selectionType) {
    case 0:  return proportionalSelection(P);
    case 1:  return tournamentSelection(P);
    case 2:  return rankingSelection(P);
  }
  return P;
}

/** Scaled Fitness Proportional Selection using
    Stochastic Uniform Sampling */
private SearchPoint[] proportionalSelection
    ( SearchPoint [] P) {
  double [] Phi = new double [populationSize];
  double sumPhi = 0.0;
  double [] Pr = new double [populationSize];
  int [] num  = new int [populationSize];

  SearchPoint [] selectedPopulation = new
      SearchPoint[populationSize];
  for (int i = 0; i < populationSize; i++)  {
    selectedPopulation[i] = new SearchPoint();
    selectedPopulation[i].init();
  }

  for (int i = 0; i < populationSize; i++) {
    Phi[i] = Math.pow( 1.0/(1+P[i].Cost - xs.Cost)
        , scalingExponent) ;
    sumPhi += Phi[i];
  }

  for (int i = 0; i < populationSize; i++)
    Pr[i] = Phi[i]/sumPhi;

  num = SUS(Pr);

  for (int i = 0, j = 0; i < populationSize; i++) {
    for (int k = 0; k < num[i]; k++) {
      selectedPopulation[j].setValueOf(P[i]) ;
      j++;
    }
  }

  return selectedPopulation;
```

```
}



/** Stochastic Uniform Sampling */
private int [] SUS (double [] Pr) {
  double pVal = SearchPoint.randGen.nextDouble();
  double sum = 0.0;
  int [] num  = new int [populationSize];
  for (int i = 0; i < populationSize; i++) {
    num[i] = 0;
    sum += Pr[i];
    while (pVal < sum) {
      num[i]++;
      pVal = pVal + (1.0 /
          ( (double) populationSize ) );
    }
  }
  return num;
}


/** tournament based selection scheme
    assuming P is already sorted by ascending costs*/
private SearchPoint [] tournamentSelection
    ( SearchPoint[] P) {
  SearchPoint [] selectedPopulation = new
      SearchPoint[populationSize];
  for (int i = 0; i < populationSize; i++)  {
    selectedPopulation[i] = new SearchPoint();
    selectedPopulation[i].init();
  }

  int [] Choices = new int [tournamentSize];

  for (int i = 0; i < populationSize; i++) {
    for (int j = 0; j < tournamentSize; j++) {
      Choices[j] =
          SearchPoint.randGen.nextInt(populationSize);
    }
    Arrays.sort(Choices);
    selectedPopulation[i].setValueOf(P[Choices[0]]);
  }

  return selectedPopulation;
}


/** ranking based Selection */
private SearchPoint[] rankingSelection (SearchPoint[] P){
  double [] Phi = new double [populationSize];
  double [] Pr = new double [populationSize];
  int [] num  = new int [populationSize];
  SearchPoint [] selectedPopulation = new
      SearchPoint[populationSize];
  for (int i = 0; i < populationSize; i++)  {
    selectedPopulation[i] = new SearchPoint();
    selectedPopulation[i].init();
  }

  double alphaRank = 2 - betaRank;

  for (int i = 0; i < populationSize; i++)  {
    Pr[i] = (alphaRank + ((i+1)/(populationSize - 1))
        *2*(betaRank - alphaRank))/populationSize;
  }
  num = SUS(Pr);
  for (int i = 0, j = 0; i < populationSize; i++) {
    for (int k = 0; k < num[i]; k++) {
      selectedPopulation[j].setValueOf(P[i]) ;
      j++;
    }
  }

  return selectedPopulation;
}


/** generate offspring by mutation */
private SearchPoint[] mutate (SearchPoint[] P) {
  SearchPoint [] mutatedPopulation = new
      SearchPoint [populationSize];
  for (int i = 0; i < populationSize; i++)  {
    mutatedPopulation[i] = new SearchPoint();
```

```
      mutatedPopulation[i].init();
    }
    for (int i = 0; i < populationSize; i++)
      mutatedPopulation[i].setValueOf(P[i].neighbor(NToUse));
    return mutatedPopulation;
  }

}
```

Bibliography

[1] V.O.K. Li and X. Qiu, "Personal Communication Systems (PCS)," *Proceedings of the IEEE*, vol. 83, no. 9, pp. 1210-43, September, 1995.

[2] J.D. Gibson, Ed., *The Mobile Communications Handbook*, CRC Press, 1996.

[3] C. Muller, E.H. Magill, P. Prosser, and D.G. Smith, "Artificial Intelligence in Telecommunications," *IEEE Global Telecommunications Conference*, GLOBECOM '93, vol. 2, pp. 883-887, Nov. 1993.

[4] M.R. Garey and D.S. Johnson, *Computers and Intractability: a guide to the theory of NP-completeness*, W. H. Freeman & Company, 1979.

[5] J. H. Holland, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, 1975.

[6] S. Kirkpatrick, C.D. Gelatt jr., and M.P. Vecchi, "Optimization by Simulated Annealing," *Science*, no. 220, pp. 671-680, 1983.

[7] D. Bertsekas and R. Gallager, *Data Networks*, Prentice Hall, 1992.

[8] G. Rudolph, "Convergence Analysis of Canonical Genetic Algorithms," *IEEE Transactions on Neural Networks*, vol. 5, no. 1, Jan. 1994.

[9] M. Mitchell, *An Introduction to Genetic Algorithms*, MIT Press, 1997.

[10] D. Goldberg, *Genetic Algorithms in search, optimization and learning*, Addison Wesley, 1989.

[11] T. Bäck, D.B. Fogel, and Z. Michalewicz, *Handbook of Evolutionary Computation*, IOP Publishing and Oxford University Press, 1997.

[12] E.H.L. Aarts and J.H.M. Korst, *Simulated Annealing and Boltzmann Machines*, Wiley, 1989.

[13] F. Glover, "Tabu Search - Part I," *ORSA Journal on Computing*, vol. 1, 190-206, 1989.

[14] F. Glover, "Tabu Search - Part II," *ORSA Journal on Computing*, vol. 2, 4-32, 1989.

[15] F. Glover and M. Laguna, *Tabu Search*, Kluwer Academic Publishers, 1997.

[16] T. S. Rappaport, *Wireless Communications - Principles & Practice*, Prentice Hall, 1996.

[17] E. Aarts and J. K. Lenstra, *Local search in Combinatorial Optimization*, Wiley, 1997.

[18] K.T. Ko et al., "Using Genetic Algorithms to design mesh networks," *Computer*, vol.30, no.8, pp. 56-61, Aug. 1997.

[19] S. Pierre and A. Elgibaoui, "Improving communication network topologies using Tabu Search," *22nd Annual Conference on Local Computer Networks (LCN'97)*, pp. 44-53, 1997.

[20] G. Celli, E. Costamagna, and A. Fanni, "Genetic Algorithms for telecommunication network optimization," *IEEE International Conference on Systems,*

Man and Cybernetics, Intelligent Systems for the 21st Century, pp. 1227-1232, 1995.

[21] E. Costamagna, A. Fanni, and G. Giacinto, "A Simulated Annealing algorithm for the optimization of communication networks," *URSI International Symposium on Signals, Systems, and Eletctronics – ISSSE '95*, pp. 40-8, 1995.

[22] M. Shahbaz, "Fixed network design of cellular mobile communication networks using Genetic Algorithms," *Fourth IEEE International Conference on Universal Personal Communications*, pp. 163-7, 1995.

[23] M. Shahbaz, "Random Guided Algorithms for Fixed Network Design of Cellular Mobile Communication Networks," *Teletraffic Contributions for the Information Age. 15th International Teletraffic Congress*, vol. 1, pp. 255-64, June 1997.

[24] J. G. Proakis, *Digital Communications*, McGraw-Hill, 1995.

[25] I. Katzela and M. Nahshineh, "Channel assignment schemes for cellular mobile telecommunication systems: a comprehensive survey," *IEEE Personal Communications*, vol. 3, no. 3, pp. 10-31, June 1996.

[26] P. Raymond, "Performance analysis of cellular networks," *IEEE Transactions on Communications*, vol. 39, no. 3, pp. 542-548, 1994.

[27] W.K. Lai and G.C. Coghill, "Channel assignment through evolutionary optimization," *IEEE Transactions on Vehicular Technology*, vol. 45, no. 1, pp. 91 -96, Feb. 1996.

[28] F.J. Jaimes-Romero, D. Munoz-Rodriguez, and S. Tekinay, "Channel assignment in cellular systems using Genetic Algorithms," *IEEE 46th Vehicular Technology Conference*, vol. 2, 741-5, 1996.

[29] C.Y. Ngo, V.O.K. Li, "Fixed channel assignment in cellular radio networks using a modified Genetic Algorithm," *IEEE Transactions on Vehicular Technology*, vol. 47, no. 1, pp. 163-72, Feb. 1998.

[30] N. Funabiki and Y. Takefuji, "A neural network parallel algorithm for channel assignment problems in cellular radio networks," *IEEE Transactions on Vehicular Technology*, vol. 41, no. 4, pp. 430-437, 1992.

[31] M. Duque-Anton, D. Kunz, B. Ruber, "Channel Assignment for Cellular Radio Using Simulated Annealing," *IEEE Trans on Vehicular Technology*, vol. 42, no. 1, pp. 14-21, 1993

[32] K. Min-Jeong, "A comparison of two search algorithms for solving the channel assignment problem," *5th IEEE International Conference on Universal Personal Communications*, vol. 2, pp. 681-5, 1996.

[33] Hao Jin-Kao; R. Dorne, P. Galinier, "Tabu search for frequency assignment in mobile radio networks," *Journal of Heuristics*, vol. 4, no. 1, pp. 47-62, June 1998.

[34] A. Ortega et al., "Algorithmes de Coloration des Graph et d'Affectation des Fréquences, Approche Géométrique et Analytiques, Heuristiques, Recuit Simulé, Programmation par Contraintes, Algorithme Génétiques," *Technical Report NT/PAB/SRM/RRm/4353*, CNET, Paris, 1995.

[35] P. Calegari et al., "Genetic approach to radio network optimization for mobile systems," *IEEE 47th Vehicular Technology Conference*, vol. 2, pp. 755-9, 1997.

[36] W.K. Hale, "Frequency Assignment: Theory and Applications," *Proceedings of the IEEE*, vol. 68, no. 12, pp. 1497-1514, Dec. 1980.

[37] H.R. Anderson, and J.P. McGeehan, "Optimizing Microcell Base Station Locations Using Simulated Annealing Techniques," *IEEE 44th Vehicular Technology Conference*, vol. 2, pp. 858-862, 1994.

[38] V. Erceg et al., "Urban/suburban out-of-sight propagation modeling," *IEEE Communications Magazine*, pp. 56-61, June 1992.

[39] T.P. Wang, S.Y. Hwang, and C.C. Tseng, "Registration Area Planning for PCS Networks Using Genetic Algorithms," *IEEE Transactions on Vehicular Technology*, vol. 47, no. 3, pp. 987-994, Aug. 1998.

[40] S. Junping, and H. C. Lee, "Optimal mobile location tracking by multilayered model strategy," *Third IEEE International Conference on Engineering of Complex Computer Systems*, pp. 86-95, 1997.

[41] A. Yener, and C. Rose, "Genetic algorithms applied to cellular call admission: local policies," *IEEE Transactions on Vehicular Technology*, vol. 46, no. 1, pp. 72-9, Feb. 1997.

[42] S. Verdu, *Multiuser Detection*, Cambridge University Press, 1998.

[43] X.F. Wang, W. S. Lu, and A. Antoniou, "A Genetic Algorithm-based multiuser detector for multiple-access communications" *IEEE International Symposium on Circuits and System - ISCAS'98*, vol. 4, pp. 534-7, 1998.

[44] M.J. Juntti, T. Schlosser, and J. O. Lilleberg, "Genetic algorithms for multiuser detection in synchronous CDMA," *IEEE International Symposium on Information Theory*, pp. 492, 1997.

[45] C. J. Chang, C. H. Wu, "Optimal frame pattern design for a TDMA mobile communication system using a Simulated Annealing algorithm," *IEEE Transactions on Vehicular Technology*, vol. 42, no. 2, pp. 205-11, May 1993.

[46] M. S. White, S. J. Flockton, "A genetic adaptive algorithm for data equalization," *Proceedings of the First IEEE Conference on Evolutionary Computation*, vol. 2, pp. 665-669, 1994.

[47] S. Chen, Y. Wu, and S. McLaughlin, "Genetic algorithm optimization for blind channel identification with higher order cumulant fitting," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 4, Nov. 1997.

[48] S. Chen, B.L. Luk, Y. Liu, "Application of adaptive Simulated Annealing to blind channel identification with HOC fitting," *Electronics Letters*, vol. 34, no. 3, pp. 234-5, Feb. 1998.

[49] M.A. Alkanhal, S.A. Alshebeili, "Blind estimation of digital communication channels using cumulants and Genetic Algorithms", *International Conference on Consumer Electronics*, pp. 386-387, 1997.

[50] D.G. Luenberger, *Introduction to linear and nonlinear programming*, Addison-Wesley, 1973.

[51] W. Sacco, *Dynamic Programming : an elegant problem solver*, Janson Publications, 1987.

[52] G.L. Nemhauser and L.A.Wolsey, *Integer and Combinatorial Optimization*, Wiley, 1988.

[53] D.H. Wolpert and W.G. Macready, "No Free Lunch Theorems for Optimization," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, April 1997.

[54] D.H. Wolpert and W.G. Macready, "No free lunch theorems for search," Santa Fe Institute, Santa Fe, NM, Tech. Rep. SFI-TR-05-010, 1995.

[55] W.G. Macready and D.H.Wolpert, "What makes an optimization problem hard?" *Complexity*, vol. 5, pp. 40-46, 1996.

[56] R. Marett and M. Wright, "A comparison of Neighborhood Search Techniques for Multi-Objective Combinatorial Problems," *Computers in Operations Research*, vol. 23, no. 5, pp. 465-483, 1996.

[57] A. Moilanen, "Parameterization of a metapopulation model: an empirical comparison of several different genetic algorithms, simulated annealing and tabu search," *1995 IEEE International Conference on Evolutionary Computation*, vol. 2, pp. 551-6, 1995.

[58] M. Chakraborty, U.K. Chakraborty, "Applying Genetic Algorithm and Simulated Annealing to a Combinatorial Optimization Problem," *Information, Communications and Signal Processing, ICICS '97*, Singapore, September 1997.

[59] G. McMahon and D. Hadinoto, "Comparison of Heuristic Search Algorithms for Single Machine Schedule Problems," *Progress in Evolutionary Computation. AI'93 and AI'94 Workshops and Evolutionary Computation. Selected*

Papers, Melbourne, Vic. & Armidale, NSW, Australia November 1993 & November 1994.

[60] M. Yagiura, T. Ibaraki, "Metaheuristics as Robust and Simple Optimization Tools," *Proceedings of IEEE International Conference on Evolutionary Computation (ICEC'96)*, pp. 541-6, Nagoya, Japan, May 1996.

[61] T. Ibaraki, "Comparison with existing optimization methods," in *Handbook of Evolutionary Computation*, eds. T. Bäck, D.B. Fogel, and Z. Michalewicz, pp. D3.6:1-D3.6:2.

[62] P.M. Pardalos, K.A. Murthy, and T.P. Harrison, " A computational comparison of local search heuristics for solving quadratic assignment problems," *Informatica*, vol.4, no.1-2, p. 172-87, 1993.

[63] R.S. Freedman, R. DiGiorgio, "A comparison of stochastic search heuristics for portfolio optimization," *Proceedings of AI Applications on Wall Street*, New York, NY, USA, April 1993.

[64] K. Singh and R. Bochynek, "Comparison of heuristic search methods for sequencing problems. A computational study," *First International Conference on Operations and Quantitative Management*, vol. 2, pp. 451-8, January 1997.

[65] D.S. Johnson and L.A. McGeoch, "The Traveling Salesman Problem: A Case Study in Local Optimization", in *Local Search in Combinatorial Optimization*, eds. E.H.L. Aarts and J.K. Lenstra, John Wiley and Sons, pp. 215-310, 1997.

[66] D.S. Johnson, "A Theoretician's Guide to the Experimental Analysis of Algorithms," *http://www.research.att.com/∼dsj/papers/exper.ps*, (incomplete internet draft), April 1996.

[67] J. Grefenstette, "Proportional selection and sampling algorithms," in *Handbook of Evolutionary Computation*, eds. T. Bäck, D.B. Fogel, and Z. Michalewicz, pp. C2.2:1-C2.2:7.

[68] T. Blickle, "Tornament selection" in *Handbook of Evolutionary Computation*, eds. T. Bäck, D.B. Fogel, and Z. Michalewicz, pp. C2.3:1-C2.3:4.

[69] J. Grefenstette, "Rank-based selection," in *Handbook of Evolutionary Computation*, eds. T. Bäck, D.B. Fogel, and Z. Michalewicz, pp. C2.4:1-C2.4:6.