Mobility Management in Cellular Telephony

Benjamin P. Cooke[*], Darongsae Kwon[†], Dmitry Glotov[‡],
Simon Schurr[§], Daniel Taylor[¶], Todd Wittman[‖]

Industrial Mentor: David F. Shallcross[**]

June 3, 2002

1 Introduction

In the world of cellular telephony there is a hierarchy of controlling devices. Cellular telephones communicate with Base Transceiver Stations (BTS) or transceivers, which in turn are assigned to Base Station Controllers (BSC) or controllers. All controllers are connected to a Mobile Switching Centers (MSC). As a user of a cellular telephone moves around, the call must be transfered from transceiver to transceiver, and sometimes from controller to controller. We are interested in the problem of minimizing the cost of these transfers from controller to controller. In the above hierarchy, we consider a subtree emanating from one Mobile Switching Center. The two sets corresponding to the subtree are I, the set of transceivers, and J, the set of controllers. The problem is to assign each transceiver from the set I to a controller from the set J optimally subject to certain constraints.

We introduce $|I||J|$ binary variables x_{ij}. If the i-th transceiver is assigned to the j-th controller then $x_{ij} = 1$, otherwise $x_{ij} = 0$.

$$x_{ij} \in \{0, 1\} \quad i \in I,\ j \in J$$

We now formulate the constraints in terms of these variables.

[*]Duke University
[†]Seoul National University, Korea
[‡]Purdue University
[§]University of Maryland
[¶]Washington State University
[‖]University of Minnesota
[**]Telcordia

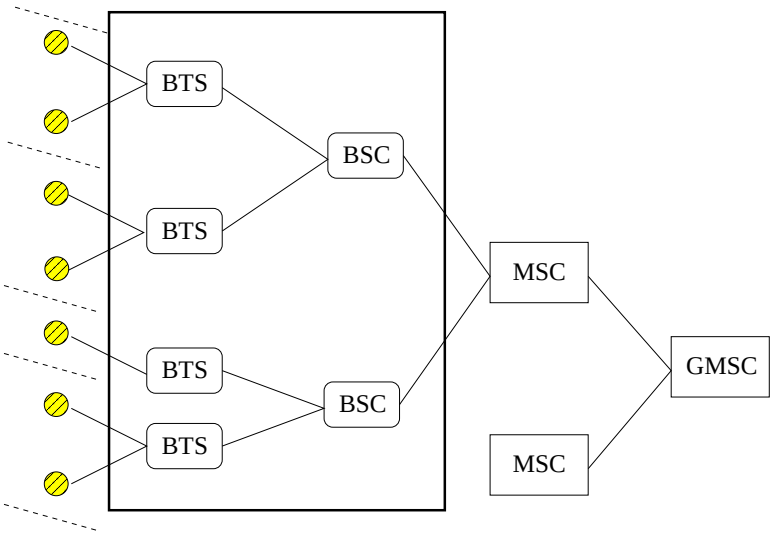

Figure 1: Cellular network

$$\sum_{j \in J} x_{ij} = 1, \quad i \in I \tag{1}$$

$$\sum_{i \in I} x_{ij} \leq n_j, \quad j \in J \tag{2}$$

$$\sum_{i \in I} t_i x_{ij} \leq c_j, \quad j \in J \tag{3}$$

$$\sum_{i \in I} a_i x_{ij} \leq d_j, \quad j \in J \tag{4}$$

$$\sum_{i,k \in I} f_{ik} \left(q x_{ij} x_{kj} + r x_{ij} \sum_{l \neq j} x_{kl} + s x_{kj} \sum_{l \neq j} x_{il} \right) \leq e_j z, \quad j \in J \tag{5}$$

The meaning of (1) is that each transceiver is assigned to exactly one controller. The second set of constraints means that the total number of transceivers attached to the j-th controller cannot

exceed n_j. The third set of constraints limits the amount of traffic that can go through a controller. Here t_i is the traffic through transceiver i and c_j is the maximum allowed traffic through controller j. The next final set of limitations is that the maximum number of call attempts a controller can handle is d_j. Here a_i represents the number of call attempts at transceiver i. (1) through (4) are all linear constraints.

The constraints in (5) are quadratic. The left hand side of each inequality (5) is composed of the load resulting from hand-offs within a controller, from a controller, and to a controller. Here, q represents the load from handing over a call between transceivers as signed to the same controller; r represents the load on the controller from handing over a call to some other controller; and s represents the load from accepting a call from some other controller. Often we will make the assumption that $q < r, s$, and when convenient we assume $r = s$. And finally, f_{ik} represents the hand-offs between transceivers i and k. It is reasonable to consider the case $f_{ik} = f_{ki}$, i.e. the flow matrix is symmetric. Our objective is to minimize the *bottleneck* of the system. i.e., minimize the maximum relative load z over all controllers. For controller j, the relative load is the left-hand-side of (5) divided by e_j, the capacity of controller j.

Hence the problem can be stated in two possible ways:

(6) minimize z subject to (1), (2), (3), and (4), where

$$z = \max_{j \in J} \frac{\sum_{i,k \in I} f_{ik} \left(q x_{ij} x_{kj} + r x_{ij} \sum_{l \neq j} x_{kl} + s x_{kj} \sum_{l \neq j} x_{il} \right)}{e_j}$$

or

(7) minimize z subject to (1), (2), (3), (4), and (5),

Throughout the paper we denote by z^* the value of the objective function corresponding to an optimal solution of (6) or (7).

2 Deriving simple a priori bounds on z^*

The quadratic constraints in (5) can be written as

$$\begin{aligned} e_j z &\geq \sum_{i,k \in I} f_{ik} \left(q x_{ij} x_{kj} + r x_{ij} \sum_{l \neq j} x_{kl} + s x_{kj} \sum_{l \neq j} x_{il} \right) \\ &= \sum_{i,k \in I} f_{ik} \Big(q x_{ij} x_{kj} + r x_{ij} (1 - x_{kj}) + s x_{kj} (1 - x_{ij}) \Big) \quad \text{in view of (1)} \\ &= \sum_{i,k \in I} f_{ik} \Big((q - r - s) x_{ij} x_{kj} + r x_{ij} + s x_{kj} \Big), \quad j \in J. \end{aligned}$$

Summing these $|J|$ inequalities and using (1) again, we have that

$$\sum_{j\in J} e_j z \;\geq\; \sum_{i,k\in I} f_{ik}\left((q-r-s)\sum_{j\in J} x_{ij}x_{kj} + r + s\right).$$

But

$$\begin{aligned} 0 \;\leq\; \sum_{j\in J} x_{ij}x_{kj} \;&\leq\; \sum_{j\in J} x_{ij} \sum_{j\in J} x_{kj} \quad \forall x_{\cdot,\cdot} \in \{0,1\} \\ &=\; 1 \qquad \text{from (1).} \end{aligned}$$

As explained previously, it is intuitively reasonable that $q < r, s$, so $q - r - s < 0$. We then have that

$$\sum_{j\in J} e_j z \geq \sum_{i,k\in I} f_{ik}\Big((q-r-s)+r+s\Big) = q\sum_{i,k\in I} f_{ik},$$

and therefore the following lower bound for z^*:

$$z^* \geq \frac{q\sum_{i,k\in I} f_{ik}}{\sum_{j\in J} e_j}.$$

Now we know that at least one of the constraints in (5) will be active at the optimal solution. (If at some point all constraints are inactive, we cannot be optimal, because we can decrease z while maintaining feasibility of (1)-(5)). Hence for any optimal solution $(x^*, z^*) \in \{0,1\}^{|I|\cdot|J|} \times \mathcal{R}_+$, there exists $j' \in J$ such that

$$\begin{aligned} z^* \;&=\; \frac{1}{e_{j'}} \sum_{i,k\in I} f_{ik}\left((q-r-s)x^*_{ij'}x^*_{kj'} + rx^*_{ij'} + sx^*_{kj'}\right) \\ &\leq\; \frac{1}{e_{j'}} \sum_{i,k\in I} f_{ik}(rx^*_{ij'} + sx^*_{kj'}) \\ &=\; \frac{r+s}{e_{j'}} \sum_{i\in I}\left(\sum_{k\in I} f_{ik}\right) x^*_{ij'} \quad \text{from the symmetry of } f \\ &\leq\; \frac{r+s}{e_{j'}}\|f\|_\infty \sum_{i\in I} x^*_{ij'} \\ &\leq\; \frac{r+s}{\min_j e_j}\|f\|_\infty\|n\|_\infty \quad \text{from (2),} \end{aligned}$$

where the matrix norm for f is induced by the ∞ vector norm.

Computational experience shows that the lower bound can be reasonably useful. However, the upper bound is very loose. This can probably be attributed to the fact that the lower bound takes

all the nonlinear constraints into consideration, whereas the upper bound is only based upon the constraint $j' \in J$.

Depending on the type of algorithm that is employed to find an near-optimal solution to (6), these bounds may be useful in filtering out some suboptimal solutions.

3 Algorithms

Four algorithms were proposed for solving this problem. The first three are probabilistic algorithms. These include a Metropolis Algorithm with annealing, a Genetic Algorithm, and a Greedy Search. The fourth algorithm, the Branch and Bound, theoretically searches through the whole space of feasible solutions and ultimately finds the optimal solution. All of these algorithms were suggested for the Quadratic Assignment Problem in [C].

3.1 Metropolis

The Metropolis algorithm [MRT] was one of the algorithms we used for the problem. The idea behind the algorithm is to search through the space of feasible solutions by making random moves, calculating the function we are trying to minimize, z, and accepting or rejecting the move based on the improvement in z.

The algorithm is comprised of the following steps. First, randomly assign each controller to a transceiver. Ensure constraints (2), (3), and (4) are satisfied. If not, continue selecting random configurations of transceivers assigned to controllers until all constraints are satisfied. When they are satisfied, calculate z and then start a repeating loop. The first step of the loop is performing a random move to a new configuration. The random move consists of randomly selecting a controller j, then randomly selecting a transceiver i assigned to that controller, and finally randomly selecting a controller k to assign to transceiver i. Figure 2 shows a small example of a random move.

After the random move has been made, the constraints are checked again. If they are not satisfied, we reject the move and return to the previous configuration. If they are satisfied, we calculate z. If the value of z for the new configuration is better than the value of z for the previous configuration, we accept the new configuration. If not, we accept the new move with a probability of

$$P(accept) = e^{\frac{-\Delta z}{T}}, \tag{8}$$

where $\Delta z = z_{new} - z_{old}$ and T is an algorithm parameter. After the move has been either accepted or rejected, we repeat the loop.

When running the algorithm, we pick many different random starting locations and lower T methodically. Because solutions that satisfy all the constraints may form disconnected regions, we need to pick many starting locations to attempt to search the entire space. Also, as we run the

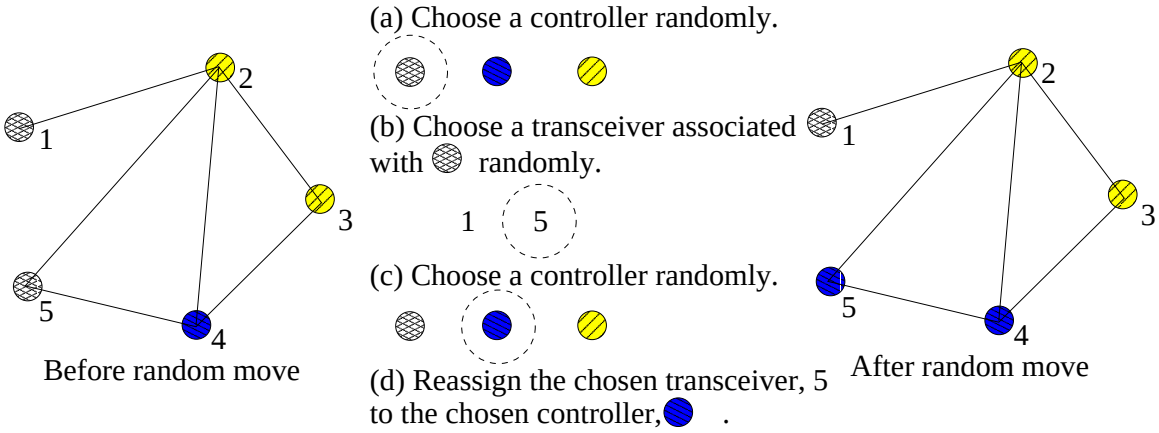

Figure 2: A random move

algorithm, we lower slowly T, lowering the probability of accepting moves that do not decrease z. The purpose of lowering T is to focus on regions where there is are local minimums and search more selectively through those regions.

When making the random move, one must make sure the move is truly random. At first, we calculated a random transceiver to switch directly for each move. This non-random move caused the algorithm to trend toward configurations with equal number of transceivers assigned to each controller. For example, consider a configuration with controllers A and B assigned 20 and 10 transceivers, respectively. Then choosing a transceiver randomly would cause a transceiver assigned to A to be picked more of the time.

The benefits of this algorithm are the speed of computation and the algorithm's ability to escape local minimums. One of the drawbacks of any random search algorithm is the inability to search the entire space in some reasonable amount of time.

During implementation of the algorithm, considerations were taken for speed of computation. In other words, large matrix manipulation was not done and rejection of bad moves was done quickly which avoided needless computation. However, constraints were only checked after the new move was computed, which is a place the algorithm could improve. The order of the constraints also might be changed for certain problems, and more efficient transition between configurations could be implemented. The main computational cost was calculating z, so other improvements would only speed up the algorithm marginally.

3.2 Genetic Algorithm

A genetic algorithm (GA) is a random search heuristic based on the theories of evolution and survival of the fittest (see [M]). Each solution in the search space is encoded as a binary string, which can be thought of as the DNA code of an organism. Each organism x has an associated fitness $F(x)$. The goal is to create a population of k organisms, "evolve" them into stronger organisms, and weed out weak ones by natural selection. More explicitly, the evolution of a population proceeds in three phases. First, a crossover phase "breeds" k new organisms. Two organisms are selected at random and they create two offspring, with each offspring receiving a gene from either parent with equal probability. Second, the "mutation" phase changes the gene in any position with some small probability p_m. This allows the algorithm to explore more parts of the solution space. The final phase is selection, which determines which of the $2k$ parents and offspring survive into the next generation of size k. In proportional selection, an organism survives with a probability proportional to its fitness. These three phases are repeated until an organism with a desired fitness is found or a maximum number of iterations is reached. A simple, or canonical, GA is described below:

Initialize a random population
Repeat until some convergence criterion is met
    Crossover
    Mutation
    Selection

In our case, a natural binary encoding is to write a solution as a vector of the x_{ij}'s. We are trying to minimize the function z in (6). Since we maximize fitness, we should set $F = 1/z$. However, we need to incorporate the constraints into the fitness to ensure that we produce a feasible solution. So we create a penalty function

$$Pen(x) = \sum_{\text{constraint } i} C \max(\ 0,\ LHS_i - RHS_i\ ),$$

where C is some positive constant and LHS_i and RHS_i represent the left and right hand sides of constraint i, respectively. This penalty function measures how many constraints are violated and to what extent. We then define our fitness function as

$$F(x) = \frac{1}{z + Pen(x)}.$$

This fitness function will force the GA to move into a feasible region quickly.

We implemented a simple GA in MATLAB. After some experimentation, we settled on the parameters $p_m = 0.01, k = |I|/2$, and $C = 10$. Generally, a 0-1 encoding is desirable to ensure convergence. However, the GA under the 0-1 encoding was sluggish in detecting a feasible region. We switched to an integer encoding, where each solution is a list of controller numbers indicating the assignment of each transceiver by position. This sped up convergence greatly because constraint (1) is automatically satisfied under this encoding. Such an encoding is usually recommended for assignment problems [DW]. To ensure that the GA did not get stuck near a local optimum, we

implemented a restart schedule. We ran the GA for a fixed number of iterations, comprising one tournament, and restarted with a random population, saving the best solution to the side. Figure 3 is a screenshot showing a restart schedule. The spikes in the graph correspond to the start of a new tournament.

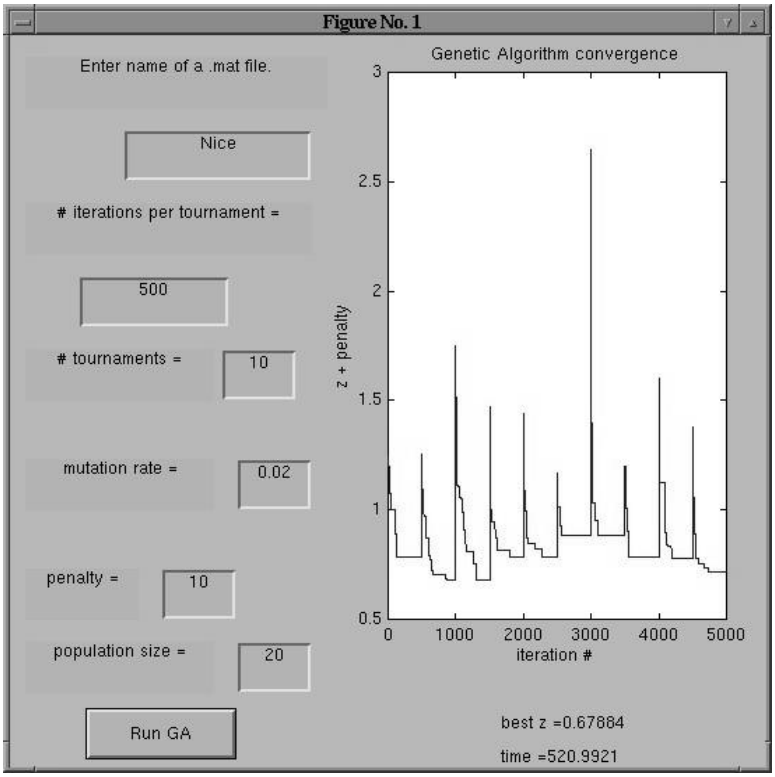

Figure 3: A typical run of the Genetic Algorithm

Although the GA is known to converge to the global optimum [R], it tended to do so very slowly for this problem. For large test cases ($|I| \sim 500$), the algorithm was unable to produce a feasible answer in a reasonable amount of time. Eranda Cela has noted that simple genetic algorithms have difficulty producing optimal or near-optimal answers for large instances of the Quadratic Assignment Problems [C]. Convergence could be improved by experimentally determining the parameters p_m and k as well as the restart schedule. One approach to correct this is to produce a so-called genetic hybrid by combining a simple GA with a local search heuristic such as tabu search or hillclimbing. Ahuja et. al. produced a genetic hybrid incorporating a greedy search that obtains near-optimal (within 1%) results for instances of the Quadratic Assignment Problem [AOT]. A first step in improving the running time would be to code the algorithm in a more efficient language, such as C/C++.

3.3 Greedy Assignment Algorithm

The quadratic semi-assignment problem (QSAP) is known to be NP-hard. Hence we face the challenge to reduce the computational time. Another probabilistic method of solving the problem is the Greedy Assignment Algorithm (GAA). The idea finds its roots in the Greedy Randomized Adaptive Search Procedure (GRASP) designed to solve QAP (see [QAP]). An appealing feature of GAA is that it requires computing only portions of the objective function rather than the value of the whole objective function at every iteration. Thus the computational time is reduced.

We repeat the assigning process for sufficiently many times as follows, saving the smallest z. During each repetition, we first assign each controllers an initial random transceiver, or a seed. We then repeat the following loop. First we calculate the 'gain' from adding each possible remaining transceiver to each controller. We choose the transceiver/controller combination, i j, that gives the best 'gain' and satisfies all the constraints. We then assign transceiver i to controller j and repeat the loop. After all transceivers have been assigned, compute the value for z.

The idea behind the 'gains' stems from the following argument. Let transceiver i be assigned to controller j. Let z_j be the left hand side of (5) for a fixed j. If a new transceiver k is assigned to controller j, then the contribution to z_j is $q(f_{ik} + f_{ki})/e_j$. Otherwise, the contribution is $(sf_{ik} + rf_{ki})/e_j$. Therefore, gains which are achieved by assigning transceiver i to controller j can be defined as follows:

$$\widetilde{Gain}(i,j) = \sum_{k \in \text{controller } j} \frac{(s-q)f_{ik} + (r-q)f_{ki}}{e_j}.$$

If $r = s > q$ and f is symmetric, we can redefine as following:

$$Gain(i,j) = \sum_{k \in \text{controller } j} \frac{f_{ik}}{e_j}.$$

The results for this algorithm presented in this paper are under the assumptions that $r = s > q$ and f is symmetric.

To compute gains achieved by assigning a new transceiver to one controller, there must be some transceivers which have already been assigned to the controller. Hence the need for random seeds to initialize each repetition of the algorithm.

In the case $|I| >> |J|$, it easily finds a good solution since the seletion of random seeds does not have a big effect on the result. On the other hand, if $|I|$ is not dominant, bad seeds can result in large z. Therefore, if $|I|$ is large and not dominant, we need many repetitions to ensure that enough different seeds have been selected.

```
smallest_z = +∞
while (repetition < limit) {
    Assign initial random seeds:
        Assign transceivers to controllers randomly
        so that each controller has one transceiver.
    Assign all the other transceivers:
        assigned_transceivers = |J|
        while (assigned_tranceivers < |I|) {
            For unassigned transceivers, construct a matrix G
            whose entries are G(i, j) = Gain(i, j).
            A = assigned_transceivers + 1
            while (assigned_transceivers < A) {
                Find (i, j) where G(i, j) is maximal.
                if (all the constraints are satisfied) then
                    assign transceiver i to controller j
                    assinged_transceivers = A
                else
                    G(i, j) = 0
            }
        }
    Calculate value for z:
    Update value of z:
        if (z < smallest_z) then
            smallest_z = z
    Increase counter:
        repetition = repetition + 1
}
```

Figure 4: Pseudo-code of Greedy Assignment Algorithm

3.4 Branch and Bound algorithm

3.4.1 Introduction

Branch and Bound algorithms give us a means for solving optimization problems containing integer variables. It is well known that the general integer programming problem is $\mathcal{NP}$-hard, but by relaxing the integrality constraints in a sense to be explained below, we can reduce the problem to that of solving several subproblems whose variables are all continuous. The first Branch and Bound algorithm for an optimization problem containing integer variables was presented in 1960 [LD], and since then, these algorithms have been successfully applied to many hard Combinatorial Optimiza-

tion problems.

To apply such an algorithm requires us to find a set of 'hard' constraints that can be relaxed. Given the difficulties posed by the integrality constraints, it seems natural to relax these to the bound constraints $0 \leq x_{ij} \leq 1, \ \forall \ (i,j) \in I \times J$. This is a relaxation in the mathematical sense because the replacement of the integrality constraints by bound constraints clearly enlarges the feasible region of (7), while the Objective Function is unchanged. It follows that feasible solutions to the relaxed problem provide lower bounds for z^*, the optimal Objective Function Value (OFV) of (7). The ability of Branch and Bound algorithms to provide lower (as well as upper) bounds on z^* is a feature that is absent in many other classes of discrete optimization algorithms (which only find upper bounds for minimization problems).

The solution of any subproblem in which the integrality constraints are relaxed to bound constraints will generally be fractional. The name 'Branch and Bound' is derived from the way in which we 'branch' on the fractional variables. When we branch on some fractional variable $x_{i'j'}$, we recognize that in the optimal solution to (7), its value will be 0 or 1. So we enforce these two additional constraints separately, hence the two branches, each of which leads to a different subproblem with fewer variables.

3.4.2 Approximating the Nonlinear subproblems by Linear subproblems

It would be ideal if it were sufficient to relax the integrality constraints in each subproblem of the Branch and Bound Algorithm. If so, each subproblem would be quadratically constrained with continuous variables. However the feasible region of (1)-(5) generally forms a nonconvex set (due to the strictly concave quadratic functions on the left-hand-side of (5)). Therefore a nonlinear solver might produce a sequence of points that converge to a local, but nonglobal, optimum. This is a serious problem since we depend on solving each subproblem to (global) optimality. (This allows us to obtain the lower bounds on z^*.) For minimization problems, the OFV associated with a local optimum solution will be at least as much as that of a global optimal solution, and hence might not be a true lower bound on z^*.

Therefore we will *lower bound* the quadratic functions in (5) by linear functions. This will have the effect of enlarging the feasible region of (7). Hence the optimal OFV of the problem with linear lower bounds in (5) will provide a lower bound on that of the quadratically constrained problem. The advantage in bounding the quadratic functions by linear functions is that the resulting subproblems are linear program (LPs), so any local optima are also global optima. The drawback is that the lower bounds obtained by solving these LPs might not be tight because of the gap between each quadratic function and its linear lower bound.

As mentioned previously, we assume $q < r, s$. Using the relation $xy \leq \frac{1}{2}(x+y)$ for any binary variables x and y, we then have the following characterization of the aforementioned linear lower

bounds: for each $j \in J$,

$$\begin{aligned} e_j z &\geq \sum_{i,k \in I} f_{ik} \Big((q-r-s)\frac{1}{2}(x_{ij}+x_{kj}) + s(x_{ij}+x_{kj}) \Big) \\ &= \frac{q}{2} \sum_{i,k \in I} f_{ik}(x_{ij}+x_{kj}) + \sum_{i,k \in I} f_{ik} \left(\frac{-(r+s)}{2}(x_{ij}+x_{kj}) + r x_{ij} + s x_{kj} \right) \\ &= \frac{q}{2} \sum_{i,k \in I} f_{ik}(x_{ij}+x_{kj}) \quad \text{in view of } f \text{ being symmetric} \\ &= q \sum_{i \in I} \left(\sum_{k \in I} f_{ik} \right) x_{ij} \\ &= q \sum_{i \in I} \|f_{i\cdot}\|_1 \, x_{ij}, \end{aligned}$$

where $f_{i\cdot}$ denotes the ith row of f.

Hence the linear inequalities $q \sum_{i \in I} \|f_{i\cdot}\|_1 x_{ij} - e_j z \leq 0, \ j \in J$ can be used in place of (5) to give a lower bound on z^*.

3.4.3 Implementation issues

Branch and Bound algorithms differ widely in flavor. In theory, we can obtain an optimal solution to an integer program via explicit or implicit enumeration of certain relaxed subproblems. However even implicit enumeration can sometimes be computationally prohibitive, and in practice we may be forced to use a stopping criterion such as "Relative gap between Best known Upper and Lower bounds < some predetermined value". Alternatively we may have to stop after a certain amount of computer running time.

In terms of deciding which variable to branch on (if a choice exists), it seems sensible to branch on a variable that is furthest from integrality (feasibility), i.e., is closest to 0.5.

A depth-first search was used to decide the order in which to solve the subproblems. Such a scheme has the advantage of trying to find an integer (nonfractional) solution as fast as possible. We can then use the resulting bound from this solution to eliminate certain feasible solutions that are known in advance to be suboptimal.

Our version of the Branch and Bound algorithm was implemented in MATLAB, and the `linprog` routine there was used to solve the linear subproblems. When invoking this routine, we had the option of using an Interior Point Method or an Active Set method to solve each LP. Although Interior Point Methods are generally regarded as being superior to Active Set methods for most types of LPs, we used the latter since they can easily take advantage of the fact that we are performing a depth-first search. (Most of the time, we only one add simple constraint – $x_{i'j'} = 0$ or $x_{i'j'} = 1$ – to each subproblem to obtain the next subproblem, so the reoptimization should be relatively inexpensive.)

3.5 Other Possible Algorithms

We considered combining the Metropolis algorithm with the greedy search algorithm. The idea was to take the best solutions generated by the Greedy algorithm and feed the solutions into the Metropolis algorithm. The reasoning was that the Greedy algorithm does not search locally so combining it with the Metropolis algorithm might lead to finding better local solutions. However, combining these two algorithms did not work because the Greedy algorithm is indeed looking for a local minimum. With the Greedy algorithm, if two starting points ever converge to the same assignment of transceivers to controllers, they will continue together from that point on. So the Greedy algorithm is in fact already looking for a local minimum.

We also explored the possiblility of combining the Genetic Algorithm with a local greedy search. This was done by Lamarckian evolution, in which the algorithm deterministically creates a fitter organism by greedily selecting which genes to mutate. However, this process requires additional computations of the fitness function and slowed down the overall running time of the algorithm.

4 Results

4.1 Data Acquisition and Generation

Due to the lack of empirical data, generated random data was determined to be sufficient for testing the algorithms covered in this survey. There were two different types of data we needed to estimate. The first type of data was the use numbers for individual transceivers, as well as the number of call attempts, a_i, the amount of traffic, t_i, and the amount of transfer between two transceivers, f_{ik}. T he maximum number of transceivers a controller can operate, n_j, maximum allowed traffic through controllers, c_j, maximum number of ca ll attempts a controller can handle, and capacity of controllers, e_j, comprise the second type of data.

According to Lam, Cox and Widom [LCW], one model for the transfer rate between two transceivers i and k, called the 'fluid model', makes f_{ik} jointly proportional to the length of the boundary between the two cells and the velocity flow. This model is based on repre senting cellular communacation by a moving fluid, with people talking on the phone and moving around being analogous to particles in the fl uid. This means that knowing what transceivers are adjacent to transceiver i is important. From information gathered from southern Fran ce, [T] we estimated the locations of the BTS's and produced a graph based upon the adjacency of the cells. This graph does not appea r to be planar. This is due to the fact that mountains, hills, or buildings may interfere with the transceivers' signal producing disconne cted cells.

In order to try and model this phnomenon, we first chose points randomly in the unit square. $(x, y) = U(0, 1)$ The Delaunay triangulation of these points was formed, to produce a model for connected cell adjacency. Additional edges were added randomly based upon their relative distance and the number of nodes.

Actual traffic data was hard to estimate, so t_i was generated randomly. Information provided

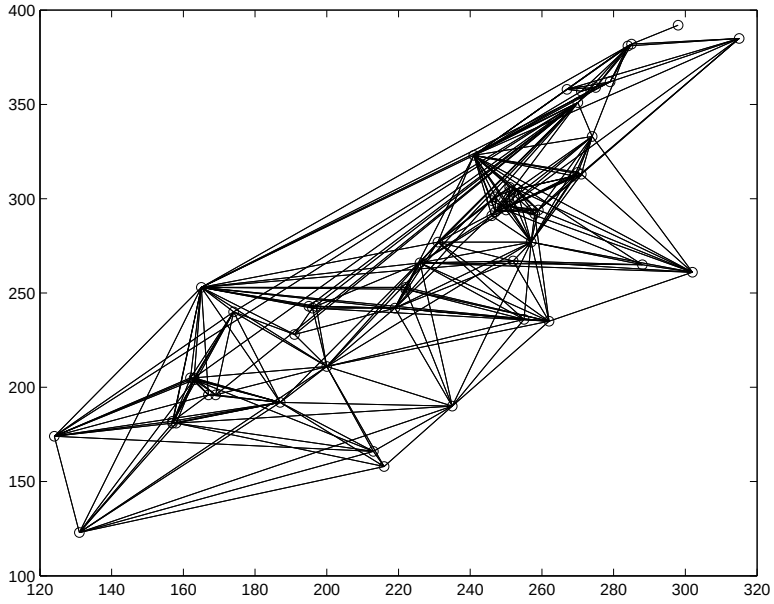

Figure 5: Telecommunications Data from Southeastern France

by our mentor said that each BTS could have multple channels provided by TRX's placed on the BTS. He also said that each channel could hold up to 8 calls simutaneously. To model this, first a random number of TRX (1 - 3) was chosen, then a normal μ=5 σ=1 random variable for the number of calls per channel. Multiplying these gave a t vector. Our mentor said estimate about 15 call attempts per hour per erlang of traffic. To get the a_i vector, a normal μ=15 σ=1.5 random variable was chosen, and multplied for each element of t. This was the estimate for the a vector. The matrix f_{ij} was estimated by averaging the traffic in each cell and multiplying it by a uniform (.25,1.25) random variable. This number was to estimate the length of the boundary between the cells and the velocity of flow.

The second type of data we needed to estimate was the capabilities of the controllers. In out model represented by the n_j, c_j d_j and e_j. We were given a paper on the specifications of a controller produced by Lucent Technologies [L]. One of our mentor's associates [E] told us that a realistic size problem was 500 - 800 transceivers and 6 - 10 controllers. This gives us about 80 transceivers per controller. For our models the Lucent data was scaled to the transceiver per controller ratio.

4.2 Quantitative Results

To test the three random search algorithms, we generated both sparse and dense random data sets. The sparse sets had f matrices that contained roughly 10

For the small data sets, the Metropolis algorithm outperformed the Genetic Algorithm and

	Metropolis			Genetic			Greedy		
$\|I\|$	20	50	100	20	50	100	20	50	100
z=1.0	0	.002	.004	10.708	3.2335	7.1738	0	.012	.04
z=0.9	0	.002	.004	10.708	3.2335	7.1740	0	.012	.04
z=0.8	0	.002	.004	10.708	3.2335	7.1741	0	.012	.04
z=0.7	0	.002	.004	10.708	3.2335	7.1741	0	.012	.04
z=0.6	0	.002	.004	10.708	3.2335	7.1742	0	.012	.04
z=0.5	0	.002	.004	10.708	3.2335	7.1742	0	.012	.04
z=0.4	0	.008	.004	253.7684	9.1346	9.9672	0	.012	.04
z=0.3	.002	517.552	334.1425*				0	.012	.04
z=0.2	38.212								8.67
Best z	0.1801	0.2457	0.2786	0.3499	0.3469	0.3332	0.2230	0.2374	0.1900

Table 1: $T(z)$ for Sparse Data Sets

Greedy Algorithm. However, the Greedy Algorithm detected the best solutions quickly for larger data sets. Each of the three algorithms seemed to perform better on the sparse data sets. Sparse data has fewer constraints to check, so it is reasonable to expect the algorithms to detect feasible solutions faster. It should be noted that the Metropolis and Greedy Algorithm were coded in C++, whereas the Genetic Algorithm was written in MATLAB. Despite the discrepancy between the programming languages and hence the running times, the Metropolis and Greedy algorithms detected better solutions with fewer computations and hence outperformed the Genetic Algorithm.

5 Conclusions and Recommendations

The Branch and Bound method becomes too expensive computationally for large problems, but it is still a useful tool for developing bounds on the optimal solution. A lower bound on the optimal solution will provide a random search heuristic with a much-needed stopping criterion. Of the three random search methods we implemented, the Greedy Algorithm performed the best in both speed and accuracy.

None of the algorithms was able to produce feasible solutions for a very large test data set ($|I| = 500$), which is probably representative of the actual size of the problem for a metropolitan area. The algorithms had difficulty processing the f matrix, which is $|I|$ by $|I|$. We recommend that sparse matrix computation techniques be incorporated into the algorithms. Also, it may become necessary to implement parallel processing to solve large instances of the problem.

*Only four of the five trial runs reached this z value.

	Metropolis			Genetic			Greedy		
$\|I\|$	20	50	100	20	50	100	20	50	100
z=1.6	.002	.016	.028	4.7423	19.3386	39.2149	0	.008	.088
z=1.5	.002	.016	1.962	4.7423	19.3386	111.2582*	0	.008	.088
z=1.4	.002	.016		4.7423	19.3386		0	.008	.316
z=1.3	.002	.016		4.7423	19.3386		0	.008	35.1375*
z=1.2	.002	.016		4.7423	19.3386		0	.008	
z=1.1	.002	.016		4.7423	19.3386		0	.008	
z=1.0	.002	.016		4.7423	19.3386		0	.008	
z=0.9	.002	75.968		4.7426			0	.036	
z=0.8	.002			4.7426			0		
z=0.7	.002			4.7427			0		
z=0.6	.818			82.1572*			0		
Best z	0.5494	0.8330	1.4066	0.5752	0.9031	1.4924	0.5636	0.8498	1.2733

Table 2: $T(z)$ for Dense Data Sets.

Acknowledgements

We would like to thank the Institute for Mathematics and its Applications and the University of Minnesota for the opportunity to participate in the Workshop for Graduate Students: Mathematical Modeling in Industry held at the IMA from May 26 to June 3, 2002. We would also like to thank our mentor Dr. David Shallcross and his colleagues at Telcordia for their support and guidance.

References

[AOT] R.K. Ahuja, J.B. Orlin, and A. Tivari. A Greedy Genetic Algorithm for the Quadratic Assignment Problem. *Working paper* 3826-95. Sloan School of Management, MIT, 1995.

[C] Eranda Cela. The Quadratic Assignment Problem: Theory and Algorithms. *Kluwer Academic Publishers*, Dordrecht, 1998.

[DW] John Dzubera and Darrell Whitley. Advanced Correlation Analysis of Operators for the Traveling Salesman Problem. *Lecture Notes in Computer Science*, Vol. 866: Parallel Problem Solving from Nature - PPSNIII, pages 119-129, 1994.

[E] Personal e-mail correspondence between David Shallcross and Eric Van den Berg, Telcordia

[L] Lucent Technologies, Base Station Controller Frame (BCF-2000), *http://www.lucent.com/livelink/157602_Brochure.pdf*

[LCW] Lam, Derek; Cox, Donald C; Widom, Jennifer. Teletraffic Modeling for Personal Communication Services. *http://www-db.stanford.edu/pub/papers/teletraffic.ps*

[LD] A.H. Land and A.G. Doig, An Automatic Method for Solving Discrete Programming Problems, *Econometrica* 27, pp497-520, 1960

[M] Zbigniew Michalewicz. Genetic Algorithms + Data Structures = Evolution Programs. *Springer-Verlag*, Berlin, 1992.

[MRT] Metropolis, N., Rosenbluth, A., Rosenbluth, M., Teller, A., and Teller, E. 1953. Equations of state calculations by fast computing machines. *Journal of Chemical Physics*. 21, 1087-1092.

[QAP] Quadratic Assignment and Related Problems, Pardalos + Wolkowicz, eds., *DIMACS* vol 16, American Methematical Society, 1994

[R] Gunter Rudolph. Convergence Analysis of Canonical Genetic Algorithms. *IEEE Transactions on Neural Networks*, 5(1): 96-101, January 1994.

[T] Telecommunications data from the southeast of France, *Telcordia internal documentation*

[W] L.A.Wolsey, Integer Programming, *John Wiley & Sons*, 1998