**Heuristic Methods for Channel Allocation in Cellular Networks**

Michael Jalkio

Cornell University, Computer Science, [mrj77@cornell.edu](mailto:mrj77@cornell.edu)

Kevin Li

Cornell University, Computer Science, [kyl27@cornell.edu](mailto:kyl27@cornell.edu)

Daniel Sperling

Cornell University, Computer Science, [dhs252@cornell.edu](mailto:dhs252@cornell.edu)

Abstract- In this paper we explore different heuristic methods for optimization in the channel allocation problem for cellular networks. We used random sampling, simulated annealing, tabu search, a genetic algorithm, and a modified dynamically dimensioned search (DDS). Among these algorithms simulated annealing found the best results. However, modifying SA to be a stochastic greedy search led to even better results. We then developed a custom heuristic (which we call the Sperling method) which outperformed all the other heuristic methods. Our overall best result led from running the stochastic greedy search with the result from the Sperling method as the initial solution.

# Introduction

In this assignment we are fictitiously working as design engineers for Big Red Wireless, Inc. They’re a cellular service start-up, and have been allocated 50 cell channels by the FCC. They must allocate these channels amongst 7 cells which have an average of 166 users. The average traffic vector is:

The co-channel and co-cell constraints can be modeled with the interference matrix **I**:

In the matrix, the diagonal elements model the co-cell constraints. If channel *c* is allocated to cell *j*, then the next channel that can still be allocated to *j* is channel *c +* **I**(*j,j*). Otherwise, we have violated that co-cell constraint. For the cellular network being used by Big Red Wireless we see that all of the co-cell constraints require we leave at least one unused channel between channels allocated to a given cell.

The co-channel constraints are all of the non-diagonal elements of **I**. If **I**(*j,k*) = 0, the same channel can be used in cells *j* and *k*. If **I**(*j,k*) = 1, the two cells must use different channels.

Our goal is to allocate the channels to minimize the number of customers who will experience issues with our cellular network. Therefore, we wish to minimize the number of constraints which are violated.

# Decision Variables and Search Space

A potential channel allocation is represented by a 7x50 matrix **A**. An element **A**(*j,k*) = 1 if channel *k* is allocated to cell *j* and is 0 otherwise. We restrict the search space to only valid allocations, where the *j*th row of **A** sums to **T**(*j*).

The size of the search space is therefore:

So there is absolutely no way that we could search the entire search space.

# Cost Function

The cost function works by considering the channel allocations in **A**. If there are two channel assignments *a1* = **A**(*j1,k1*) and *a2* = **A**(*j2,k2*) where both *a1* and *a2* are 1 we say that there is a co-cell constraint violation if

And a co-channel constraint if

The number of these constraints that are broken for a given allocation **A** is its cost, and we seek to minimize the cost.

# Algorithm Descriptions

## Random Sampling (RS)

The title of the paper is centered 25mm (1”) below the top of the page in 16 or 14 point bold font. Use 14 pt font only if the length of your title makes it necessary.

## Simulated Annealing (SA)

The simulated annealing algorithm is a modification on a stochastic greedy search. The weakness of stochastic greedy search is that it can easily get “stuck” in a local optimum if its neighborhood is too small. SA avoids this by having a probability of accepting a solution which is less good than the current optimal solution. Over time it reduces this probability, and gets closer and closer to the stochastic greedy search.

## Tabu Search (TS)

After implementing the base Tabu search algorithm, we determined that the neighborhood space for each solution was much too large to search exhaustively. Therefore, we experimented with different ways of constraining the neighborhood search, while preserving an even search distribution.

We modified the base Tabu search algorithm such that neighbors of the current solution are computed by shuffling a fixed number of bits in a given cell (row). Each shuffle iteration is defined as a single pairwise swap of two randomly selected bits in the cell. The number of shuffle iterations used to compute each neighbor is a parameter which we define at the beginning of each trial. For each cell, we compute a fixed number of neighbors each having been perturbed in that cell. A cell is defined as tabu if it was perturbed in the recent past, thus preventing searching neighbors with perturbations in the same cell.

## Genetic Algorithm (GA)

The title of the paper is centered 25mm (1”) below the top of the page in 16 or 14 point bold font. Use 14 pt font only if the length of your title makes it necessary.

## Dynamically Dimensioned Search (DDS)

Dynamically dimensioned search was left relatively unchanged from the original implementation provided for homework 8. A neighbor of the current solution is computed by selecting a subvector of each cell (row vector) to shuffle, where the size of the subvector follows a gaussian distribution with standard deviation sigma. This subvector would be selected from a random location anywhere within the cell. Effectively, each cell is treated as a separate dimension for DDS to search. After each iteration, the probability of changing each dimension decreases exponentially.

# The Sperling Method

The main text for your paragraphs should be 10pt font with 1 or 2 pt leading. Again, don’t use a typewriter-style font (e.g. Courier). Use a more readable and clear font like Times Roman for your paper.

All body paragraphs should have the first line indented about 4.4mm (.175”) except for the first paragraph follow­ing a heading which is not indented.

# Running Algorithms and Results

## Random Sampling (RS)

The title of the paper is centered 25mm (1”) below the top of the page in 16 or 14 point bold font. Use 14 pt font only if the length of your title makes it necessary.

## Simulated Annealing (SA)

In simulated annealing multiple parameters can be set: initial temperature, alpha, beta, and M. For this application we chose to only vary alpha and the initial temperature, setting beta to 0 and M to 1 in all trials.

Before experimenting with parameter values we needed to define our neighborhood. For SA, we found neighbors by randomly selecting a cell, and then changing one of its channel allocations. So within a row in a current allocation **A**, we set a random value from 0 to 1, and another value from 1 to 0.

We began our analysis of SA using parameters that had been used to solve another problem. These values were:

The average best cost with these parameters was 166.93, and the standard deviation was 2.84. Since these values were completely naïve, we knew that we could do better.

We then calculated the average change in our cost function for a solution and its neighbors, which was found to be 3. We decided that we wanted the probability of making an uphill move to be 90% initially, and for it to be 50% after 500 iterations. Using this we calculated new parameter values that were more fit to the problem:

Using this led to an average best cost of 108.33 and standard deviation of 3.02, a huge improvement! However, looking at the data it appeared that the algorithm was making the most improvements in the greedy stages (after 500 iterations). We ran a number of trials and discovered something very upsetting…setting the parameters so that SA became a stochastic greedy search led to better results than the true SA! With parameters:

We achieved an average best cost of 93.43, with a standard deviation of 3.33. We believe the reason for this is the absolutely massive neighborhood for SA to utilize. With such a large neighborhood it’s almost impossible to get caught in a local minimum, so it doesn’t make sense to choose any solutions with worse costs. It’s better to be greedy and to just wait until SA finds new optimal solutions.

Across all trials the average CPU time was approximately 1.25 seconds.

## Tabu Search (TS)

The base Tabu Search algorithm took approximately 30 seconds per *iteration*, since a single pairwise swap of two (different value) bits in one cell would constitute a new neighbor. The number of neighbors for a given solution is on the order of several thousand. This basic approach was much too slow to compute any meaningful solutions. Therefore, we decided to experiment with different ways of constraining the neighborhood search space for each current solution.

By limiting the number of pairwise swaps per cell, and the total number of neighbors having perturbations in each given cell, we were able to reduce the per-trial time to approximately 30 seconds. Better results could have been achieved with longer computation time and larger neighborhood searches. However, the marginal increase in average best solution cost was not worth the extra cpu time required.

Parameters that needed to be set include: tenure length, number of swaps per neighbor, number of neighbors for each cell. We determined that each of these parameters had little impact on the overall performance (relative to computation time) of Tabu search. This is likely due to the vast search space and limited number of solutions that could be found within the limited computation time.

With tenure length as 2, 2 neighbors for each cell, and 5 swaps per neighbor, we achieved an average best cost of 165.8, with a standard deviation of 1.6.

## Genetic Algorithm (GA)

The title of the paper is centered 25mm (1”) below the top of the page in 16 or 14 point bold font. Use 14 pt font only if the length of your title makes it necessary.

## Dynamically Dimensioned Search (DDS)

DDS provided results similar to the other algorithms. The one parameter that needed to be set in DDS was the standard deviations (sigma’s) of the Gaussian perturbations on each cell. Experiments were done with a wide range of sigma’s, but the best results were produced when sigma was large. That is, when the cell (row) was shuffled randomly in its entirety. This represents a degeneration of DDS into Random search, which had nearly the same performance.

We were able to achieve an average best cost of 163.4, with a standard deviation of 2.56, and an average CPU time of 1.75 seconds.

# Comparison of Algorithms

Do not use footnotes in your paper.

# Best Solution

Author’s name and year, e.g., (Fogel 1995), is the preferred format. Check to be sure that the references are complete and accurate.

The example bibliography format is given in the Bibliography section. Be sure to list the page numbers for articles appearing in edited vol­umes, such as conference proceedings. Also be sure to list the first initials and full last names of the editors for an edited volume.

# Conclusion

The problem of channel allocation for cellular networks is very difficult to solve due to the sheer size of the solution space. Conventional optimization methods cannot be applied to such a problem due to limitations on execution time. Thus, we apply heuristic approaches to search for as a good of a solution as possible within the given time constraints. We have tested several heuristic algorithms including Random Sampling, Simulated Annealing, Greedy Search, Tabu Search, and Dynamically Dimensioned Search. We have also evaluated and compared the results produced by these algorithms, as well as the results produced by our own novel proprietary algorithm: The Sperling MethodTM. From our analyses, Simulated Annealing outperformed all widely used traditional heuristic algorithms. What came as a surprise was discovering that as we made Simulated Annealing greedier, it kept finding increasingly better results. However, The Sperling MethodTM produced even better results than Simulated Annealing, and is vastly less complex than any of the other algorithms. Thus, from a practical standpoint, the problem of channel allocation for cellular networks is most effectively solved using basic human intuition, with minimal assistance from conventional heuristic search algorithms.

Team Member Contributions

Michael Jalkio modified the simulated annealing code from homework 2 for this assignment. He then tested it to find the best parameters (using random allocations as well as the Sperling method). He also wrote the beginning of the report (from the title through section 3), and the parts in section 4 and 6 dealing with SA. For funsies he ran a greedy SA for a really long time and found that it could find an allocation with a cost of only 75!

Kevin Li made modifications to the Tabu search and DDS code from previous homework assignments, and he also wrote the Random search code. He ran experiments on these algorithms logging and comparing their performance relative to CPU time. He wrote the report sections on Tabu search and DDS, as well as the conclusion section.