Heuristic Methods for Channel Allocation in Cellular Networks

Michael Jalkio
Cornell University, Computer Science, mrj77@cornell.edu

Kevin Li
Cornell University, Computer Science, kyl27@cornell.edu

Daniel Sperling
Cornell University, Computer Science, 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.

1 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:

$$\boldsymbol{T} = [32\ \ 26\ \ 14\ \ 32\ \ 18\ \ 20\ \ 24]$$

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

$$\boldsymbol{I} = \begin{pmatrix} 2 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 2 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 2 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 2 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 2 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 2 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 2 \end{pmatrix}$$

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.

2 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 jth row of **A** sums to **T**(j).

The size of the search space is therefore:

$$\binom{350}{166} = \frac{350!}{166!\,(350-166)!} \approx 6.2 * 10^{103}$$

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

3 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

$$j1 = j2 \text{ and } |k2 - k1| < \boldsymbol{I}(j1, j1)$$

And a co-channel constraint if

$$j1 \neq j2 \text{ and } |k2 - k1| < \boldsymbol{I}(j1, j2)$$

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

4 Algorithm Descriptions

4.1 Random Sampling (RS)

The random sampling algorithm was the simplest algorithm used for this task. In order to write the random sampling algorithm, we simply created a new, completely random allocation for every new value tested, and kept track of the best results over all of the trials. Random sampling does not attempt to use any past information to influence the newly found allocations.

4.2 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.

4.3 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.

4.4 Genetic Algorithm (GA)

The Genetic Algorithm was similar to the one used in the original implementations from past homeworks, with modifications done on crossover and mutation to stay within the constraints of the cell allocation space. In order to produce a random child from a given pair of parents selected by tournament selection, crossover was performed "cell-wise", i.e. a crossover between two parents would include the first n cells of one of the parents and the remaining 7-n cells of the other parent. In this fashion the constraints were maintained and the possibly useful cells from the parents were maintained.

Mutation was similarly modified to maintain the constraints. In past binary examples, mutation was performed by randomly "flipping" each bit from a 0 to a 1 or a 1 to a 0 with probability based on p_mutation. In order to ensure our allocation always maintained the required allocation, for each bit that was randomly changed, we randomly chose another bit within the same cell that had the other value. These two values were then swapped instead of either value being flipped by itself. This maintained the overall allocation of all of the cells.

4.5 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.

5 The Sperling Method

Another major modification we made was to write a specific, separate algorithm to produce a strong initial solution very quickly. Then, this initial solution was used as a starting value particularly for SA to test if a better solution could be found from that point. We focused on using this as an initial solution for SA because of the previous success we had already had with SA.

The "Sperling Method" as this algorithm was named (by Mike) involved a form of a greedy heuristic to attempt to produce cells that were at a relatively low level of conflict with themselves and each other, particularly for a network allocation. Essentially, there are two important elements to this allocation: each cell is initialized in a form such that its intra-cell conflicts are minimized (which can be trivially performed), and allocations are shifted to attempt to minimize inter-cell conflicts.

In order to minimize intra-cell conflicts, all allocations of channels are initially set to be exactly the minimum distance apart from each other such that these channels do not interfere with each other. This causes a cost of zero for that particular cell. If there are too many allocations such that it is impossible to fit all allocations in the cell without conflict, remaining slots are then filled, one allocation per space at a time to limit conflicts within the cell.

Once each cell's allocation has been decided, this particular allocation may be shifted, inverted, or spread (if possible) before adding the cell to the total allocation based on inter-cell conflicts with other cells. For the case of this particular assignment, cells were assigned greedily, with early cells filling any positions, and later cells modified to have less conflicts with earlier cells. This often produced a series of cells that were offset from each other, so as to completely eliminate conflicts between them.

Upon codifying a simple version of this algorithm specifically for this problem, the algorithm produced a basic

solution with cost 83. This would prove to be significantly better than any algorithm's results using random initial starting solutions. As such it was ultimately used as an initial solution for SA in the runs that produced the best solution.

6 Running Algorithms and Results

For the purposes of consistency, all algorithms were run to have a total of 1500 cost evaluations per trial. This allowed for more accurate comparisons of the results of the algorithms. All algorithms were also run for a total of 30 trials, giving an ample space for statistically significant comparison.

6.1 Random Sampling (RS)

Random sampling did not require any parameter setting, as all of the new elements produced are simply set randomly. As such, the process of running random sampling was simply ensuring that each trial ran for 1500 random solutions.

From the 30 trials run for random sampling, the average best cost found was 163.6, with a standard deviation of 1.54. The average CPU time across trials was approximately 1.76 seconds.

6.2 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:

$$T_0 = 152934534.3$$
$$\alpha = 0.996658$$

The average best cost with these parameters was 166.9, 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:

$$T_0 = \frac{-avg\Delta cost}{\ln(P_1)} = \frac{-3}{\ln(0.9)} = 28.47$$

$$\alpha = \left(\frac{-avg\Delta cost}{T_0 \ln(P2)}\right)^{1/G} = \left(\frac{-3}{28.47 * \ln(0.5)}\right)^{1/500} = 0.996$$

Using this led to an average best cost of 108.3 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:

$$T_0 = 0.1$$
$$\alpha = 0$$

We achieved an average best cost of 93.4, 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.

6.3 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 1.60.

6.4 Genetic Algorithm (GA)

The genetic algorithm had a number of parameters to be set: the probability of mutation occurring per bit, the probability of crossover occurring per child, and the population size vs. the number of iterations performed to give an even number of cost evaluations. Parameter setting for GA was performed by performing a minimal GA to a limited range with each of many of the parameters set to different value in turn (i.e. trying out the algorithm with a probability of mutation at 0.05 and a probability of crossover at 0.90 with a population of size 50). After finding the best set of values by trial, a smaller search was made of the values closest to the found probabilities to give a more accurate set of best parameters without incurring a huge cost. Ultimately, the best parameters found were 0.844 for the crossover probability, 0.076 for the mutation probability, and a population size of 50.

With these parameters, the GA was run in exactly the same fashion in previous homework and as described above. Ultimately, GA had an average best cost of 163.6, with a standard deviation of 1.99 and an average CPU time of 1.16 seconds.

6.5 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.

7 Comparison of Algorithms

The disparity between the algorithms that worked best and those that didn't work well can be seen in a graph of the average best cost over time for each graph (figure 1). Over the course of the 1500 iterations, simulated annealing had easily the strongest performance, with the average best cost decreasing throughout the iterations and becoming better than the final values of GA, DDS, RS, and TS after around 600 iterations.

Tabu search performs well originally, and then appears to stop finding better solutions nearly as well quickly. This appears in the plateau formed by TS at a cost a few values above the other algorithms.

Among all remaining algorithms the results appear to be very similar toward the end. Interestingly, the GA algorithm appears to perform better during the earlier iterations before plateauing, while SA does not perform particularly well early on.

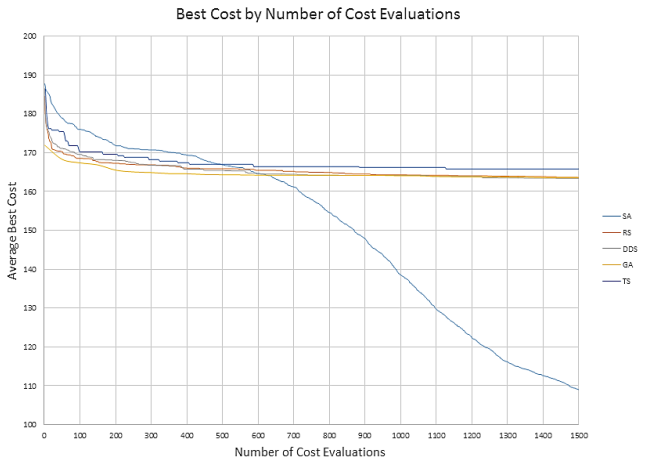

Figure 1: Average best cost of the solutions by iteration.

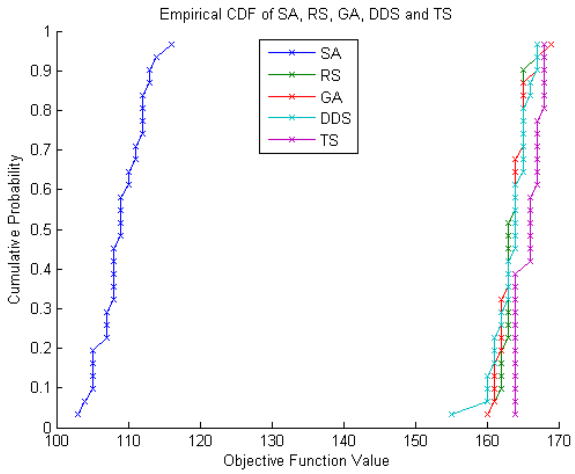

Figure 2: Empirical CDF of the best solutions

Another figure that can be used to demonstrate the relative performance of the algorithms against each other is the empirical CDF (Figure 2). This confirms many of the views seen on the best cost graph. Simulated annealing clearly stochastically dominates all of the other algorithms, and the race was not close. Each of RS, GA, and DDS have similar enough results that none of them dominate each other. This is as was expected from the best cost graph as well.

Interestingly, TS is dominated by both DDS and RS, and although it isn't *technically* dominated by GA (as there is crossover on one of the points, it is very nearly dominated by GA as well. This confirms what was seen in the average best cost graph, which is that Tabu search did not perform as well as any of the other algorithms on this particular problem.

For the purposes of completeness, a box plot is also included (figure 3).

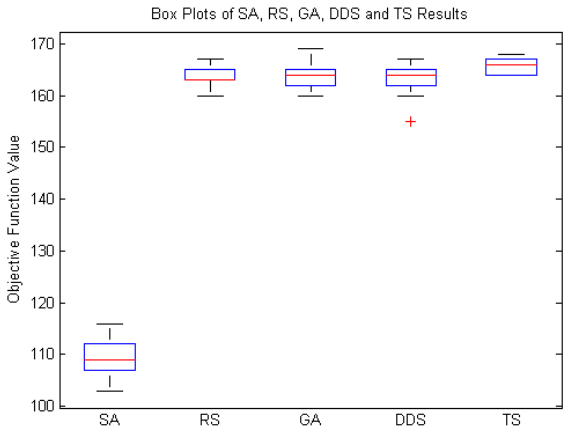

Figure 3: Boxplot of the best solutions

The most interesting pieces of information that can be gleaned from this box plot in comparison to the other visualizations are the ranges of values on a few of the algorithms. We can see that DDS actually had one particularly good best solution in the mid 150's, which is noticeably better than all of the solutions from all the other algorithms apart from SA. We also can see that the high-cost range of GA extends noticeably higher than the range for RS and DDS, which further explains why we could not conclude that GA dominated TS in the previous analysis.

Further analysis can be performed using hypothesis testing, with a null hypothesis H_0 that the means of any two of the algorithms are the same. We will use the two-sample t test based on the random allocation of starting positions among trials (used to be able to have easier parallel runs on separate computers) to attempt to disprove this hypothesis and accept an alternate hypothesis, that the found means of one or the algorithm is better than the other, demonstrating that one algorithm is better than the other.

Performing the two-sample t-test between SA and any of the other algorithms gives amusing, if expected, results. Since the mean of SA is *much* lower than the means of the other algorithms compared to the standard deviation found for each algorithm, the t-statistics found for comparison in each instance is less than -70. The test for comparison is whether $t < -t_{alpha/2,\ v}$ or $t > t_{alpha/2,\ v}$, with alpha = 0.05. As the comparison t value is approximately 2 in each instance, -70 < -2, and we reject the null hypothesis that the two means could have possibly been the same. We accept that the mean are different, and a separate hypothesis test against $t_{alpha,\ v}$ can easily allow us to accept the alternate hypothesis that the mean of SA is less than the mean found from any other algorithm. These are found up to a probability of at least 99.9999%, indicating that even for an alpha of 0.000001, the two means would be found distinct.

Comparisons between any pair of DDS, GA, and RS reveals that we cannot reject the null hypothesis that any pair of means are equivalent. The relevant t-values found were all between $0 < t < 0.5$. As again the reference $t_{alpha/2,\ v}$ is approximately equal to 2 for each pair for alpha = 0.05, we cannot reject the null hypothesis that each pair of means is the same, as t is not greater than $t_{alpha/2,\ v}$ or less than negative $t_{alpha/2,\ v}$. This confirms what we had seen visually again – the three algorithms had similar enough results that we cannot say with certainty that any one performed better on the problem than another.

When comparing any of DDS, RS, or GA with TS, we find that we can in fact conclude that each of DDS, RS, and GA performed better than TS. The relevant t statistics found were all in the range of -4 to -6. Since again the relevant value for comparison is at approximately 2, we can reject the null hypothesis that the two means were equal. Then, we can perform a separate, one-sided t-test with $t_{alpha,\ v}$ of around 1.5, and determine that each of DDS, RS, and GA had a lower mean than TS by accepting the new alternative hypothesis H_a to that affect.

All told, our comparison demonstrates that SA performed the best by far, DDS, RS, and GA each performed about equivalently well as each other (but much worse than SA), and TS performed worst on the problem, with values slightly higher than those of DDS, RS and GA.

8 Best Solution

Ultimately, the best solution found was using SA, using the cost 83 solution found from the Sperling algorithm as the initial solution. From that starting point, a solution of cost 79 was found, and that solution, A*, is depicted graphically below in figure 4.

```
                                    Channel
     1 [1010101110111111110101010101010111011101011010101]
     2 [1101010101010101010101010101010101010101010100101101]
     3 [0010100010100000100010101000001000101010101000000]
Cell 4 [1111111010101010101010101010101010101011111111011]
     5 [0101001001001001010101001001001000000101010010]
     6 [0000010101010101010101010101010101010100000100]
     7 [1010101010101010101010101000101010101010101001]
```

Figure 4: Best solution A*, cost(A*) = 79

From this best solution, it can be seen that with this particular problem, attempting to allocate 166 values into 50 channels over 7 cells, even with a strong heuristic there remain a large number of conflicts. A cost of 79 indicates that there were 79 conflicts among the 166 "users," which means that a significant proportion of the average users would be in conflict with each other, which is not good.

We were also tasked with finding a zero-cost allocation from the best solution by revoking the allocation to one member of each conflict until no conflicts remained. A naïve approach would be to simply revoke 79 of the conflicting values – 1 per each conflict – leaving a mere 87 allocations out of 166. This would not be optimal, however, because a number of the allocations are in conflict with

multiple other allocations, and a better method would be to remove allocations that were causing more conflicts. To do this, we wrote a simple successively remove the most conflicting allocation from the allotment until no conflicts remained. The result of this approach follows in figure 5.

```
                                    Channel
     1  [10101010101010101010101010101010101010101001010001]
     2  [00000101010101010101010101010101010101010100000100]
     3  [00101000101000001000101010000010000101010100000000]
Cell 4  [10101010101010101010101010100010101010101010101001]
     5  [01010000000000000000000000001000000000000001010010]
     6  [00000101010101010101010101010101010101010100000100]
     7  [10101010101010101010101010001010101010101010101001]
```

Figure 5: No-cost solution, with 132 allocations.

By using this approach, we ultimately only needed to revoke 34 of the 166 allocations to handle the 79 conflicts, indicating that there definitely were many allocations with multiple conflicts. Interestingly, many of the removed allocations were from cell 5, with only 6 of the original 18 allocations remaining. This is likely because cell 6 had the greatest level of interference with the other cells, being in next to every other cell except for cell 1. As it stands, 132 users can be supported by this system.

9 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 Method. 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 Method 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 fun he ran a greedy SA for a really long time and found that it could find an allocation with a cost of only 75! This was not included in the paper as it was run significantly longer than the other algorithms.

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.

Daniel Sperling wrote the cost function, programmed the GA code from previous assignments, and developed the "Sperling Algorithm" that found the low-cost solution to be used as an initial solution for SA, as well as an algorithm to find a zero-cost solution from that allocation. He also produced all of the statistical comparisons and graphs of the data. He wrote the sections on comparing the algorithms, the best solution found, and on GA and RS.