Automatic Frequency Assignment for Cellular Telephones Using Constraint Satisfaction Techniques[1]

Mats Carlsson, Mats Grindal
Swedish Institute of Computer Science (SICS)
PO Box 1263, S-164 28 KISTA, Sweden
{matsc,matsg}@sics.se

Abstract

We study the problem of automatic frequency assignment for cellular telephone systems. The frequency assignment problem is viewed as the problem to minimize the unsatisfied soft constraints in a constraint satisfaction problem (CSP) over a finite domain of frequencies involving co-channel, adjacent channel, and co-site constraints. The soft constraints are automatically derived from signal strength prediction data. The CSP is solved using a generalized graph coloring algorithm. Graph-theoretical results play a crucial role in making the problem tractable. Performance results from a real-world frequency assignment problem are presented.

We develop the generalized graph coloring algorithm by stepwise refinement, starting from DSATUR and augmenting it with local propagation, constraint lifting, intelligent backtracking, redundancy avoidance, and iterative deepening.

Key Words: frequency assignment, constraints, graph coloring, intelligent backtracking, iterative deepening.

1 Introduction

Since 1968, there has been a lot of interest in automatic frequency assignment for cellular telephones, but few successful implementations have been reported. Hale's article [**?**] contains an excellent overview and bibliography of the early attempts.

This paper is the result of a study in automatic frequency assignment initiated by Ericsson Radio Systems AB (ERA). The frequency assignment problem consists in assigning a number of frequencies to each of a set of base stations while minimizing the potential interference. The number of frequencies assigned varies with the required capacities of the base stations.

Currently, the construction of frequency plans for base station sites is performed manually, a job which is very time consuming. The quality of a

[1]Proc. Tenth International Conference on Logic Programming, MIT Press, pp. 647–665

man-made solution also varies, since there exist no exact algorithms for performing this task. Instead the quality is much dependent on the experience and skill of the person making the assignment.

The frequency assignment problem is viewed as the problem to minimize the unsatisfied soft constraints in a constraint satisfaction problem over a finite domain of frequencies involving three kinds of constraints:

co-channel constraints: some pairs of sites x and y must not use the same frequency ($1 \leq |c(x) - c(y)|$); these constraints are *soft*;

adjacent channel constraints: some pairs of sites x and y must not use adjacent frequencies ($2 \leq |c(x) - c(y)|$); these constraints are *soft*;

co-site constraints: a minimal separation m between frequencies f and g used at the same site is prescribed ($m \leq |f - g|$); these constraints are *hard*.

To make the problem tractable, the domain is often made up of frequency *groups* instead of individual frequencies. For example, a configuration with 12 frequency groups could be defined such that group $0 \leq i \leq 11$ contains the frequencies $\{f | f \bmod 12 = i\}$. With this definition, frequency groups 0 and 11 must be considered adjacent, i.e. modulo arithmetic must be used in the constraints. This will be tacitly assumed in the rest of the paper.

It has been noted by many authors (e.g. [**?**, **?**, **?**]) that the frequency assignment problem amounts to a graph coloring problem if only co-channel constraints are involved. Graph coloring is perhaps the most famous and most intensively investigated and applied optimization problem. It is known to be NP-complete [**?**], so the existence of a polynomial algorithm for optimally coloring general graphs seems unprobable. However, a vast number of heuristic algorithms have been suggested over the years.

Zoellner and Beall [**?**] were among the first to attack the frequency assignment problem by reducing it to a generalized graph coloring problem. Several authors (e.g. [**?**, **?**, **?**]) have since followed the same approach. Typically, the co-channel and adjacent channel constraints were derived from data about the geographic distribution of base stations and from heuristic rules. The goal was usually to find an assignment requiring the minimum number of frequencies.

In this work, we take a somewhat different approach. We derive soft constraints from predictions about signal strength data computed for each base station. Each constraint gets associated with a weight, reflecting how severe the interference would be if that particular constraint is unsatisfied in the proposed solution. Then we solve the problem of finding an assignment that minimizes the maximum weight of the unsatisfied constraints. This assignment is found by solving a series of generalized graph coloring problems on subgraphs whose constraints have a weight above a threshold value.

During the course of this study, several graph coloring algorithms have been tried with various success. The algorithm that shows the most promising behavior is the DSATUR algorithm. It generalizes readily to handling adjacent channel and co-site constraints.

The paper is organized as follows. First some basic graph theoretical terminology, definitions, and results are stated. In Section **??** the reduction from frequency allocation into graph coloring is described. Section **??** presents our framework for automatic frequency assignment. In Sections **??**, **??** and **??** the three components of the system, i.e. the compiler, the iteration shell and graph coloring algorithms are presented in greater detail. Section **??** includes an investigation of the graph coloring algorithm that we chose to focus on i.e. DSATUR, together with some general techniques that can be used to improve the performance of this type of algorithm. In Section **??** we present some test results, which are based on a real frequency assignment problem. Finally, Section **??** concludes this study.

2 Preliminaries

A *graph* G is an ordered pair consisting of a finite set $V(G)$ of *vertices* and a set $E(G)$ of unordered pairs (u, v) of distinct vertices, called *edges*. Two vertices u and v are *adjacent* if $(u, v) \in E(G)$. $N(v) = \{u \in V(G) | (v, u) \in E(G)\}$ is the *neighborhood*, $\deg(v) = |N(v)|$ the *degree* of v. If $e = (u, v) \in E(G)$ then vertex v is said to be *incident* to edge e.

G' is a *subgraph of* G if $V(G') \subseteq V(G), E(G') \subseteq E(G)$.

A *clique in* G a subgraph Cl of G with $(u, v) \in E(Cl)$ for all $u, v \in V(Cl)$. Cl is called *maximal* if there is no clique Cl' with $V(Cl) \subset V(Cl')$ and *maximum* if no clique Cl'' exists with $|V(Cl'')| > |V(Cl)|$. The *clique number* $\omega(G)$ denotes the cardinality of a maximum clique in G.

A *k-coloring* of the graph G is an assignment of an integer color value $1 \leq c(v) \leq k$ to each vertex $v \in V(G)$ such that adjacent vertices receive different color values. $\text{satdeg}(v) = |\{c(u) | u \in N(v)\}|$ is the *saturation degree* of v.

The *chromatic number* $\xi(G)$ is the minimum k for which G has a k-coloring. The determination of $\xi(G)$ for arbitrary G is known to be NP-complete [**?**]. However, the following well-known inequality provides a lower bound on $\xi(G)$:

$$\xi(G) \quad \geq \quad \omega(G) \tag{1}$$

Upper bounds on $\xi(G)$ are provided by finding feasible colorings. Matula's smallest-last coloring [**?**] can be used for this purpose:

1. **[Initialize.]** $j \leftarrow |V(G)|, H \leftarrow G$.

2. **[Find minimum degree vertex.]** Let v_j be a vertex of minimum degree in H.

3. **[Delete minimum degree vertex.]** $H \leftarrow H - v_j, j \leftarrow j - 1$.

4. **[Terminate.]** If $j \geq 1$ go to step 2, otherwise terminate.

The sequence $v_1, \ldots, v_n$ thus obtained is a *smallest-last* ordering for G. A *smallest-last* coloring of G and an upper bound on $\xi(G)$ is obtained by

$$c(v_i) \stackrel{\triangle}{=} \min\{m | 1 \leq m \neq c(v_j), v_j \in N(v_i), j < i\} \quad (2)$$

$$\delta(G) \stackrel{\triangle}{=} \max\{c(v_i)\} \quad (3)$$

$$\xi(G) \leq \delta(G) \quad (4)$$

3 The Model

Traditionally, cellular telecommunications systems are described using a model with hexagons or *cells*, with one hexagon per transmitter. The hexagonal shape was chosen mainly because it is possible to cover an area completely with equally sized hexagons. The hexagon model is however not very well suited for performance calculations, since the model does not take into account the geographical structure of the area covered nor the local properties of the transmitters. Nevertheless, many authors have developed frequency assignment methods based on a regular hexagonal cell pattern; see e.g. [**?**, **?**].

A model that more correctly takes into account the geographical structure and the local properties of each transmitter is the model used by ERA for evaluating the performance of a system. The area of the system is covered with a fine-meshed grid. In each grid point there is information on the predicted signal strength of each transmitter in the system. The cell of a transmitter is then defined as the area which contains all the grid points where that particular transmitter has the strongest signal. One implication of this definition is that a cell may consist of a number of disjoint parts.

In the grid model, the performance of the system can be measured in a particular grid point by calculating the difference in signal strength of the transmitter serving the cell and the combined signal strengths of all other transmitters using the same or adjacent frequencies.

Using the difference-method described above as a base for automatic frequency planning would be too costly, in terms of processing power. In a computation, frequencies would be allocated to transmitters depending on how much interference is caused in the rest of the system. This means that the signal quality of all grid points in all cells using a particular frequency would have to be recalculated as soon as that frequency or an adjacent frequency is tried in a new cell.

However a simplified version of the difference-method could form a base for automatic frequency planning. The system is transformed into a graph $\langle V, E\rangle$ by associating the vertices V with the transmitters and the edges E with the constraints. Every vertex is connected by edges to all other vertices, thus forming a complete graph. The edges are labelled using the information in the grid system.

There are two labels, $l_1 > l_2$, on each edge (u, v), denoting the respective weights of the co-channel and adjacent channel constraints on u and v. Edges where $l_1 = 0$ can be dropped. Co-site constraints are considered hard constraints, and our algorithm never considers solutions that violate them. Finally, the set of available frequencies or frequency groups corresponds to the set of colors available.

The weights are calculated using the following definitions:

$$f(v_i, x) \triangleq \text{ signal strength of transmitter } v_i \text{ in grid point } x \quad (5)$$

$$g(v_i) \triangleq \{x | \forall j \neq i . f(v_i, x) \geq f(v_j, x)\} \quad (6)$$

$$\alpha(v_i, v_j) \triangleq \{x | x \in g(v_i) \& f(v_i, x) - f(v_j, x) < T_{cc}(x)\} \quad (7)$$

$$l_1(v_i, v_j) \triangleq \max\left(\frac{|\alpha(v_i, v_j)|}{|g(v_i)|}, \frac{|\alpha(v_j, v_i)|}{|g(v_j)|}\right) \quad (8)$$

$$\beta(v_i, v_j) \triangleq \{x | x \in g(v_i) \& f(v_i, x) - f(v_j, x) < T_{cc}(x) - \gamma\} \quad (9)$$

$$l_2(v_i, v_j) \triangleq \max\left(\frac{|\beta(v_i, v_j)|}{|g(v_i)|}, \frac{|\beta(v_j, v_i)|}{|g(v_j)|}\right) \quad (10)$$

where $g(v_i)$ defines the cell served by transmitter v_i; $\alpha(v_i, v_j)$ determines the grid points in the cell served by transmitter v_i with co-channel interference from transmitter v_j greater than the threshold value T_{cc}; γ is the adjacent frequency attenuation (a constant > 0); and $\beta(v_i, v_j)$ determines the grid points in the cell served by transmitter v_i with adjacent channel interference from transmitter v_j greater than the threshold value. Different threshold values may be chosen for different grid points, thus tolerating less interference in grid points corresponding e.g. to highways, than in others, e.g. in unpopulated areas.

Within this model the interference $I(c)$ in the system modelled as the graph $\langle V, E\rangle$ can be defined as:

$$\begin{aligned} I(c) \triangleq \max(& \{l_2(v_i, v_j) | (v_i, v_j) \in E, |c(v_i) - c(v_j)| = 1\} \cup \\ & \{l_1(v_i, v_j) | (v_i, v_j) \in E, c(v_i) = c(v_j)\}) \end{aligned} \quad (11)$$

where modulo arithmetic may be applied in computing $|c(v_i) - c(v_j)|$. That is, given an assignment c of frequencies over the set of transmitters, the maximum interference of the whole system is defined as the maximum weight of all unsatisfied co-channel and adjacent channel constraints. Finally, an assignment c is *optimal* if it minimizes $I(c)$.

Figure 1: An overview of the system

It follows from this definition that the combined interference on a cell from several transmitters is approximated by the interference from the strongest interferer. Whether this approximation causes any problems in practice remains to be seen.

4 The Framework

The system for automatic frequency assignment consists of three modules. Figure **??** shows an overview of the system. The first module is a compiler to make the transformation from the grid representation of the system into the graph representation. The second module is a shell surrounding the graph coloring module, which is the third module. The function of the iteration shell is to supply the coloring module with an appropriate subgraph of the original graph. The coloring module then determines if the subgraph is colorable or not. The following sections will describe in more detail each of the three modules.

The purpose of splitting the system into modules is twofold, most important being time savings. The output from the compiler does not have to be regenerated unless the transmitter configuration has changed, and in that case the grid representation also has to be updated thus producing new input to the compiler.

The other reason for splitting the system into modules is for testing different algorithms without having to rewrite the whole system. Both the iteration shell and the graph coloring module offer great opportunities for exploring different ideas.

5 The Compiler

The compiler is a 250 line C program. The input to the compiler is a number of *prediction grid* files, one for each transmitter in the system, and

one *composite grid* file, containing information of which transmitter has the strongest signal for each grid point, i.e. defining the cells according to the description above. In a prediction grid there is one entry for each grid point. An entry consists of information to identify the grid point, but also the signal strength of that transmitter in that grid point.

The compiler calculates the interference between each pair of transmitters according to the formulas (**??**) and (**??**) above. For each pair of transmitters with an interference larger than zero an entry is created in the output file. An entry in the output file consists of the vertex pair and the calculated weight and type of the label (1 or 2).

6 The Shell

Prolog was chosen as the implementation language for the iteration shell and graph coloring algorithms.

The objective for the shell is to find the largest colorable subgraph of the input graph by adding the edges in descending weight order, given a specified number k of colors. When a computation is started edges with two labels i.e. l_1 and l_2 are split into two identical edges, one with each label. Similarly the neighborhood $N(v)$ of v is split into two sets, $N_1(v)$ being the vertices incident to edges with labels l_1, and $N_2(v)$ being the vertices incident to edges with labels l_2.

The shell orders the labelled edges in ascending weight order, enumerating them $1 \ldots n$. A binary search of the ordered edge list is performed until a maximal suffix is found, such that the corresponding subgraph is k-colorable. The interference in the system is defined as the largest weight of a label of any edge which is not part of the suffix.

Let G_p denote the subgraph of G that contains the edges $p+1 \ldots n$. The goal of the shell is to find the minimum m such that G_m is k-colorable. The following iteration is used:

1. **[Initialize.]** Set $l \leftarrow 0, h \leftarrow n$.
2. **[Terminate.]** If $l = h$ stop with $m = l$.
3. **[Color subgraph.]** Set $j \leftarrow \lfloor (l + h)/2 \rfloor$. Color the graph G_j. If a k-coloring is found, let $h \leftarrow j$. Otherwise, let $l \leftarrow j + 1$. Go to step 2.

Several optimizations were added to the iteration shell:

- When a coloring c has been found, it can happen that c is a feasible coloring for some subgraph $G'_j, j' < j$ (trivially recognized by checking each edge $j' + 1 \ldots j$). In this case, we set $h \leftarrow j'$ before proceeding to step 2.

- It is sometimes possible to provide a lower bound l for m by detecting a priori that $\xi(G_{l'}) > k, l' < l$. In [**?**], such lower bounds are derived, the most important one being $\omega(G_{l'})$. Other lower bounds derived from adjacent channel constraints and their combination with co-channel constraints can also be derived (Lemmas 7 and 9 of [**?**]), but have not yielded any better lower bounds than $\omega(G_{l'})$ in our experiments.

  In step 1, we initialize l to the smallest l' such that $\omega(G_{l'}) \leq k$. Computing $\omega(G_{l'})$ is itself an NP-complete problem, so this computation is only done once when the graph is loaded into the iteration shell, and could in principle be done by the compiler.

- Similarly we can obtain upper bounds h for m by detecting a priori that $\xi(G_{h'}) \leq k, h' > h$. This can be done if the smallest-last coloring of $G_{h'}$ is a k-coloring. In the presence of adjacent-channel constraints, we must use a modified definition of degree while computing the smallest-last ordering:

$$\deg(v) \stackrel{\triangle}{=} |N_1(v)| + 3|N_2(v)| \tag{12}$$

  where the factor 3 is motivated by the fact that an adjacent channel constraint on (u, v) given $a = c(u)$ implies $c(v) \notin \{(a-1) \bmod k, a, (a+1) \bmod k\}$.

  In step 1, we initialize h to the smallest h' such that $\delta(G_{h'}) \leq k$. Computing $\delta(G_{h'})$ can be done in $O(|V| + |E|)$ time [**?**]. This is also done only once.

7 Graph Coloring Algorithms

The problem of coloring the vertices of a graph in such a way that two adjacent vertices never have the same color has been of interest for mathematicians for nearly 150 years [**?**]. It is well known that this problem is NP-complete [**?**]. This means that the algorithm proposed by Brown in [**?**] and improved in [**?**] and [**?**] that solves the problem exactly is of little practical interest since it will be too time consuming. Several heuristic algorithms have however been proposed over the years, as well as various techniques for improving those algorithms. The following sections within this section will give a brief survey of some of those algorithms, and the next section will study in greater detail one of those algorithms.

One general coloring algorithm is the *sequential coloring based on order* O [**?**, **?**]:

- determine an order $O : v_1 < v_2 < \cdots < v_n$ of the vertices.
- $c(v_i) \leftarrow \min\{m | 1 \leq m \neq c(v_j), v_j \in N(v_i), j < i\}$

Needless to say, the coloring obtained will depend on the chosen order O. Matula's smallest-last coloring is a typical example.

Other sequential colorings construct the order dynamically, while coloring the vertices. One such strategy is known as *Recursive Largest First RLF* [**?**]. In RLF, a current color is determined, then the uncolored vertices are scanned one by one. If it is possible to assign the current color to the vertex, this is done and that vertex is removed from the set of uncolored vertices. If, on the other hand, the assignment would violate the rules for coloring, the vertex has to wait until the next scanning. When all vertices have been scanned with one color, a new color is chosen to be current color and the still uncolored vertices are scanned again. This procedure is then repeated until either all vertices are colored, and hence the coloring succeeded, or there are no colors left, in which case the coloring failed.

The big advantage of the RLF algorithm is that it is very fast. However, the quality of the solution is sensitive to the order in which the vertices are scanned. Sorting the vertices by degree, i.e. the number of edges leading to a vertex, and starting the scan at the vertex with the highest degree, yields a solution with a fair quality at a relatively small cost. Other improvements are possible.

There also exist a number of methods that are not sequential colorings; see [**?**] for an overview and bibliography.

8 The DSATUR Family of Algorithms

8.1 Introduction

Yet another sequential coloring based on a dynamic vertex order is known as *DSATUR* [**?**]. The following steps are performed until all vertices have been colored:

- pick out a vertex v_i with maximal saturation degree.
- choose a color for v_i among the feasible ones for that vertex.

There are two major reasons why we became interested in using this algorithm as a base for further development. The first reason is that the quality of the solution is slightly better than the one for the modified RLF, and the execution times are shorter. What is more important, however is that DSATUR in the original version, offers big possibilities for improvement.

The first improvement is choosing a data structure which optimizes the selection of the vertex with maximal saturation degree. This is accomplished by keeping an explicit representation of the *domain* of feasible colors for the vertex. Whenever a color a is chosen for v, a is deleted from the domains of all vertices in $N_1(v)$, and $(a-1) \bmod k$, a, $(a+1) \bmod k$ are deleted from the domains of all vertices in $N_2(v)$.

Finding a vertex with maximal saturation degree is accomplished by using a priority queue [**?**], where each entry in the priority queue has direct pointers to its neighborhood. When there are several vertices with the same saturation degree, the tie is broken using the smallest-last order.

A further modification that improves the quality of the solution is to improve the choice of color for the selected vertex. A good idea is to choose from the domain the color that will reduce the domains of the neighbors the least. This strategy is known in the literature as the *principle of maximum overlap of denial areas* [**?**].

The following subsections will describe some techniques, independent of the algorithm, that we also explored in order to improve the performance of the DSATUR algorithm, namely: *local propagation*, *constraint lifting*, *intelligent backtracking*, and *iterative deepening*. Furthermore an important optimization for graph coloring, *redundancy avoidance*, was used.

8.2 Local Propagation

Local propagation is a *consistency technique* [**?**] to maximize the pruning of the search space by propagating the domain reductions as far as possible. The idea of local propagation is that when the domain of a vertex v is reduced, a check is made to see if there is only one color left in its domain. If that is the case, v is colored immediately, reducing the domains of its neighbors. The local propagation continues until no more domains can be reduced.

Furthermore if $N_2(v)$ is non-empty, local propagation is sometimes possible if up to three colors are left in the domain of a vertex. Consider, for example, a domain $\{a, a+1\}$. No matter which of the two possibilities is eventually chosen, it is easy to see that neither a nor $a+1$ are feasible colors for vertices in $N_2(v)$.

Local propagation will reduce the number of iterations, since more than one vertex may be colored within the same iteration. More importantly, it will prune the search space and detect failures earlier. Furthermore fewer updates of the priority queue are needed.

Local propagation as described here in fact implements *arc-consistency*, or more precisely the AC-3 algorithm, which is at the heart of the implementation of Constraint Logic Programming (CLP) languages over finite domains [**?**].

8.3 Constraint Lifting

Constraint lifting is a technique to improve the quality of the solution. The key idea of the technique is to extract implicit information that exists in the system and incorporate this information explicitly into the data structures, by means of reduced domains and/or adding new edges. This added information may contribute to making better decisions when choosing a color by

pruning the search space.

In our implementation, as soon as the domain size is less than or equal to five, all different alternatives are tried one by one by performing local propagation as described above. If some property holds for all alternatives tried, this property is added to the system. We infer the following properties:

- domain reduction, if the same element was deleted from some domain in all alternatives, and
- co-channel (adjacent channel) constraints, if two domains were disjoint by at least one (two) colors in all solutions and the new constraint was not subsumed by existing constraints.

The reason for limiting the search for common properties among five choices or less is pragmatic. The probability of finding a common property among a number of choices decreases rapidly with the number choices, while the computation time increases. We found no case where any information could be gained by applying constraint lifting to a domain with size greater than five.

To give an example, assume for instance that a vertex v after an iteration has two remaining colors a and b available. Then before exiting that iteration, a is first assumed to be the color of v and this is propagated through the system. All changes to the system are remembered and compared to the changes caused by the propagation of b. In our example assume that both a and b caused a third color c indirectly to be removed from the domain of a vertex w. Then c can be permanently removed from the domain of w, since no matter what color v will get in the future, this will result in doing this removal.

8.4 Intelligent Backtracking

Intelligent backtracking [**?**] is a technique to reduce the computation time. As suggested by the name, intelligent backtracking is an improved variant of backtracking. When using intelligent backtracking the computation may backtrack over several choice points, upon a failure, even if there are several alternatives remaining at the skipped choice points. The decision of how far to backtrack may be based on various data.

Our approach to making backtracking intelligent is to keep track of all variables involved in a conflict with remaining alternatives, backtrack to the point in the computation where the last of those variables was assigned, and try to make a new assignment to that variable.

In the DSATUR algorithm, conflicts are detected while updating the domains of adjacent vertices. When the domain of a vertex becomes empty, there is a conflict. The variables involved in that conflict are primarily all colored neighbors of that vertex. Changing the color assignment of one of

them may solve the conflict. However, changing the color assignment of another vertex, further away from the conflict, could also solve the conflict.

In our implementation of intelligent backtracking, each vertex x is associated with an initially empty *constrainer set*, i.e. the set of colored vertices with alternatives remaining that directly or indirectly caused a domain change in x. An indirect domain change is the case when the constrainer caused a direct domain change of a vertex that was subsequently colored, causing a direct or indirect domain change of x.

When a conflict is detected, the most recently colored member c of its constrainer set s is selected as the *culprit*. The state of the program is restored to a point immediately before the culprit got its last color, that color is removed from the domain of the culprit, and all members of $s\backslash\{c\}$ are added to the constrainer set of c as in [**?**]. A new color is selected and the computation continues searching for a solution.

The search space may still be too large despite the use of intelligent backtracking. One way of solving this problem is to place a restriction on the *backtracking depth*, i.e. how far the computation is allowed to be backed up when a conflict is detected.

The order in which vertices are colored is maintained by having a counter incremented at each iteration. This information can also be used for placing a restriction on the backtracking depth. If the considered vertex got its color too early, the algorithm terminates with failure. The maximum backtracking depth is 1 initially, and is increased in successive calls to the shell (see Section **??**).

8.5 Redundancy Avoidance

Another important observation in order to save time while backtracking is not to explore redundant solutions. This is mentioned in [**?**] as an important optimization for graph coloring. When coloring a new vertex, the allowable colors are the already used colors and, if yet available, one new color. Considering other unused colors will only result in redundancy.

Avoiding redundant solutions is accomplished in the program by masking away all unused colors except one from the domains of the vertices. This prevents the program from backtracking to more than one forbidden color.

The presence of adjacent channel and co-site constraints makes the above described technique not directly applicable to our application, since it may actually cause solutions to be lost. Therefore we use a modified version: The allowable colors, when coloring a new vertex, are all colors that have been removed from any domain, plus one more color. We conjecture that this causes no loss of solutions.

Another well-known redundancy avoidance technique used in graph coloring is to find a large clique and deterministically color its vertices before any other vertices are colored. Unfortunately, this technique can also cause loss of solutions in the presence of adjacent channel and co-site constraints.

We have experimented with coloring initial cliques generated by adjacent channel and co-site constraints before coloring other vertices, but this has generally led to results poorer than using the normal DSATUR heuristic.

8.6 Iterative Deepening

Iterative deepening is a technique where successive calls are made to an algorithm, increasing the backtracking depth for each call. We have incorporated this technique into the DSATUR algorithm resulting in two different versions: the "naive" and the "sophisticated" version. The latter uses information about failed attempts to solve subgraphs.

In the naive version, each call to the iteration shell increases the backtracking depth by one. Of course the time consumption increases as the search space is enlarged, as there are more possibilities for finding a solution.

However, information about failed attempts to solve subgraphs may be accumulated and used to avoid doing redundant work. In particular, we can detect how much the backtracking depth must be increased in order to make any progress on such failed attempts.

In the sophisticated version, in any call to the iteration shell with backtracking depth d, the search for an optimal position in the edge list is restricted to the interval $[p', p)$ where p is the best position so far and p' is the smallest integer such that no subgraphs $G_i, i > p'$ are known to require backtracking depths greater than d to succeed.

Ideally, the backtracking depth necessary for solving a given subgraph G_i should be non-increasing with increasing i. Unfortunately, this is not always the case, and so the above iterative deepening strategy tends to get temporarily stuck at positions where a subgraph G_i cannot be solved at backtracking depth d but some other subgraph $G_j, j < i$ can. Nevertheless, preliminary results show that using this improvement can give substantial time savings over the naive version.

9 Test Results

This section contains test results from two real frequency assignment problems on an actual cellular telephone system.

In the first problem, all cells were assigned one frequency group. In the second problem, some cells were assigned two frequency groups f, g with co-site constraints $|f - g| = k/2$. Data for the two problems:

1. 157 vertices, 2868 co-channel constraints, 465 adjacent channel constraints, 0 co-site constraints, and $k = 12$. For this problem a clique of size 13 was found at position 1381 in the edge list. A solution at position 1382 was found and must therefore be optimal.

2. 229 vertices, 6016 co-channel constraints, 934 adjacent channel constraints, 72 co-site constraints, and $k = 24$. For this problem, no cliques of size greater than 22 were found, and it is not known whether the best solution found is optimal.

All timings were obtained by running SICStus Prolog compiling to native code on a SUN 4/60. Three DSATUR variants were used with "naive" iterative deepening:

plain uses intelligent backtracking and redundancy avoidance;

plain + LP is the plain version augmented with local propagation;

plain + LP + CL is the plain version augmented with local propagation and constraint lifting.

Figure **??** shows the relation between backtracking depth, time consumption, and quality for the three variants of the algorithm run on the 12 colors problem. This problem has an optimal solution with interference 0.0566. As can be expected, "plain + LP + CL" reaches the optimum at the smallest backtracking depth, followed by "plain + LP" and "plain" (see bottom plot). Time consumption became excessive at backtracking depths greater than 21 for "plain + LP + CL" (see middle plot).

One interesting observation is that there exist some points where an increase in the backtracking depth actually leads to a speed up (middle plot). These points have in common the fact that together with the speed up, there is also an increase in the quality of the solution. Most likely, some subgraph was not colorable at the smaller backtracking depth but succeded at the greater depth, and the subsequently tried subgraphs at the smaller depths required much more time than did the subsequently tried subgraphs at the greater depth.

The top plot shows for a given interference the DSATUR variant that produces a solution with the desired quality in the shortest amount of time. For this problem, the "plain + LP" variant displays the best performance, except for very high interferences where "plain" is somewhat faster. The more powerful "plain + LP + CL" is much too slow to be competitive on this benchmark.

Figure **??** shows the same relations as figure **??** but for the 24 colors problem. The best solution found has interference 0.0263, but it is not known whether this solution is optimal. Again, "plain + LP + CL" reaches the optimum(?) at the smallest backtracking depth (bottom plot), followed by "plain + LP" and "plain", whereas "plain + LP" has the best time-interference performance, except for finding the optimum(?) where "plain + LP + CL" is competitive in the 24 colors case (top plot).

10 Conclusions and Future Work

We studied the problem of automatic frequency assignment for cellular telephones. The frequency assignment problem was viewed as the problem to minimize the unsatisfied soft constraints in a constraint satisfaction problem (CSP) over a finite domain of frequencies involving co-channel, adjacent channel, and co-site constraints. The novel idea used in this work was to automatically derive the constraints from signal strength predictions. The CSP was solved using a generalized graph coloring algorithm. Graph-theoretical results played a crucial role in making the problem tractable. Performance results from a real-world frequency assignment problem were presented.

Prolog was a rather natural choice of implementation language. Language features that proved especially useful for this application include: automatic backtracking (including that of destructive updates) to support search argorithms, compilation on the fly of specialized predicates for domain calculations, and the interactive, incremental programming environment that makes Prolog a convenient tool for exploring new algorithms.

The primary goal of this study was to find out if frequency assignment could be automatized using a graph coloring technique. The two initial requirements were that the quality of the solution must exceed the quality of the man-made solution, within a shorter time. Although this study is far from complete, the results so far indicate that both these goals are well within reach.

Another conclusion that can be drawn from the results is hardly surprising, namely the more complexity added to an algorithm, the better the quality of the solution at the cost of longer execution times. However, constraint lifting was only competitive in one extreme case, which suggests that it might be too general for this kind of application. A better candidate might be more specialized consistency techniques such as path consistency.

In our tests we have only experimented with a single benchmark. This is not enough to draw firm conclusions from. However this benchmark represents a real frequency allocation problem for a large city. By using this input we were able to compare our computer generated solution with the manually produced solution. The final outcome of this evaluation is yet to come, but preliminary results show that our solution will outperform the manual solution actually used.

Topics for future work include refining and parallelizing the algorithm. Since the program solves a search problem, it should lend itself very well to OR-parallelism. How to combine our intelligent backtracking scheme with OR-parallelism is however not obvious, and is an interesting area for further study.

Another interesting direction is to rewrite the program as a genuine CLP program over finite domains. Since our program uses a mixture of graph-theoretical, constraint satisfaction, heuristic, and application specific algorithms, it should constitute a real challenge in terms of performance,

generality, and flexibility for constraint language implementations.

Acknowledgements

The Mobile Telephone division of ERA instigated and supported in part this study, introducing us to the world of automatic frequency assignment, and kindly provided the benchmark data. This work would not have been possible without their help.

The authors thank Ralph Clarke Haygood and the anonymous referees for careful reading of the manuscript and suggesting many improvements.

Thanks are also due to our colleagues at SICS, who have provided a stimulating research environment.

Figure 2: The 12 colors problem. The relation between backtracking depth, time consumption, and quality for the three variants.

Figure 3: The 24 colors problem. The relation between backtracking depth, time consumption, and quality for the three variants.