A Tunable Greedy For Channel Assignment Problem in Cellular Network

A thesis submitted in fulfilment of the requirements for the degree of Master of Technology in Computer Science
by

Subhankar Ghosal
(Roll No: CS-1104)

Advisor :
Dr. Sasthi C. Ghosh

Advanced Computing and Microelectronics Unit
Indian Statistical Institute
Kolkata 700 108

June 30, 2014

Abstract

This report presents a probabilistic greedy algorithm for solving the channel assignment problem (CAP) in cellular networks. We took each call as a vertex of a complete edge weighed graph, termed as CAP graph, where an edge weight represents the minimum frequency separation needed between the calls represented by the terminal vertices of that edge. Our objective is to assign non-negative integers representing colors or frequencies to the vertices of the CAP graph such that the required span (maximum frequency - minimum frequency) is minimized while satisfying the frequency separation constraints represented by the edge weights. We begin with a probabilistic ordering of the vertices and apply frequency exhaustive strategy to color them. During the coloring, when color of a vertex exceeds the maximum color of previously allocated vertices, we apply a forced assignment phase to reduce the so far obtained span. Then we propose an iterative compression phase to further reduce the span obtained from applying the frequency exhaustive strategy with forced assignment phase. Finally we introduce a smoothing phase which will try to reduce the higher frequencies obtained by the compression phase with a view to increasing the channel utilization. This essentially helps to cope up with the short term demand fluctuation in a latter phase. It is observed that there is a tradeoff between the computation time and the resulted span. Our proposed algorithm is tunable in a sense that we can get better result by allowing more computation time. The proposed polynomial time algorithm is applied over the well-known benchmark instances and the obtained spans are measured. The obtained results show that the proposed algorithm performs better than the existing assignment strategies with respect to deviation from optimality and/or computation time. The time taken by our algorithm is less than 1.77 seconds (HP Z400 Workstation) even for the most difficult benchmark instances and thus is very much suitable where fast channel assignment is of primary importance while a marginal deviation from optimality may be tolerated.

Acknowledgement

I want first to acknowledge both of my parents for standing behind me for those year of struggle and sacrifice and hold my hand to be in this situation where I am right now. I further want to show respect to all of my teachers and specially Dr. Sasthi C Ghosh for their kind guidence to complete my work. Also I am thankful to the nice facility that ISI provide us and really thankful to the love and support which my friends and classmates specially Ratan, Gopinath and Badri had provide me companion all the time without that this work could not be fruitful.

Contents

Chapter 1

Introduction

In the modern mobile dominated era, the number of mobile communication devices is increasing day by day. A mobile station (MS) can communicate with a base station (BS) through a specific channel. Because of the large number of MS, it is almost impossible to satisfy each MS by a distinct channel as the available bandwidth is very limited. Two MS cannot communicate with the same frequency at the same time if they are not far away, because of the interference. This increases the noise hugely and cross-talk and/or breach of security may occur. To encounter this, the geographical region is split into several cells and a base station is established in each cell. The same channel can simultaneously be used by multiple base stations, if their mutual separation is sufficient enough to satisfy the interference. However, the frequencies assigned to two nearby stations must differ by certain minimum value in order to avoid the channel interference. The task of assigning frequency channels to the cells satisfying the interference constraints and using as small bandwidth as possible is known as the channel assignment problem (CAP). For simplicity, we assume that the frequencies are non-negative integers. The interference avoidance is then achieved by requiring a minimum separation between the frequencies assigned to a pair of stations. Here both the intra-cell and inter-cell frequency separation requirement are represented by non-negative integers. We regard the calls originated at a cell as the demands of that cell and we need to allocate one frequency for each such call. Traditionally the CAP is represented by a triplet (X, M, Λ) where 1) $X = \{0, 1, \cdots, n-1\}$ is the set of n cells, 2) $M = (m_{xy})$ is the frequency separation matrix where m_{xy} represents the minimum frequency separation needed between a call of cell x and a call of cell y, and 3) $\Lambda = \lambda_x$ is the demand vector where λ_x represents the demand of cell x. Here $m_{xy} = 0$ implies that cells x and y are non-interfering to each other and hence are allowed to transmit simultaneously using the same channel. Under this traditional representation of the CAP, our problem is to find a frequency assignment matrix $C = c_{xu}$ where c_{xu} represents the frequency assigned to call u of cell x ($0 \leq x \leq n-1, 0 \leq u \leq \lambda_x - 1$) such that the frequency separation constraints are satisfied. Here the frequency separation constraints are represented by $|c_{xu} - c_{yv}| \geq m_{xy}$ for all x, y, u and v except when both $u = v$ and $x = y$. The objective is to minimize the span where the span is defined as $\max\limits_{x,u} c_{xu} - \min\limits_{x,u} c_{xu}$. The CAP can be modeled as a graph multi-coloring problem on a graph where each node represents a base station and there is an edge if the corresponding base stations are interfering to each other. The edge between nodes x and y has a weight m_{xy} representing the minimum frequency separation needed between a call of cell x and a call of cell y. At each node we have to assign number of colors equal to the demand of that node. Thus CAP is basically a graph multi-coloring problem on this graph where the objective is to assign colors to the vertices satisfying the frequency separation constraints and demand vector while using as small span as possible. The CAP is known to be a NP-hard problem, because a special case ($\lambda_x = 1$ for all x; and $m_{xy} = 1$, if cells x and y are adjacent and 0, otherwise) of the problem is equivalent to the classical graph coloring problem. In our discussion, we will use the terms frequency/color, cell/base station and bandwidth/span interchangeably.

In this report, we present a new powerful technique to solve the CAP. We first represent the CAP in terms of a *CAP graph*. Then we introduce three techniques to reduce the span of the CAP represented by this CAP graph. We break the boundary of determinism and define a *probabilistic ordering* of the vertices of the CAP graph. Then we start coloring vertices of the CAP graph using the frequency exhaustive strategy following the obtained probabilistic ordering. During the coloring, when color of a vertex exceeds the maximum color of previously allocated vertices, we go for a *forced assignment* phase to reduce the so far obtained span. After all vertices are colored, we use a *compression* phase, where we choose the maximum colored vertex and try to reduce its color by increasing some of the other vertices' colors. We iteratively go on this compression phase till no further reduction is possible. At last we apply a *smoothing* phase where following the same probabilistic ordering of the vertices we put the minimum color to each vertex satisfying the already allocated vertices. We iteratively go on this phase as long as at least one vertex's color is reduced. This smoothing phase helps to increase the channel utilization. Our proposed algorithm is tunable in a sense that we can get better result by allowing more computation time. The proposed polynomial time algorithm is applied over the well-known benchmark instances and shown that it performs better than the existing assignment strategies with respect to deviation from optimality and/or computation time. The computation time taken by our algorithm is order of seconds and thus is very much suitable for solving the practical channel assignment problem.

Chapter 2

Previous literature and benchmarks

Because of the hardness of the problem, authors use deterministic, randomized, approximation, and heuristic algorithms to solve the CAP. Heuristic algorithms like genetic algorithms [8, 9, 17, 6], neural networks [22, 15], tabu search [5] and simulated annealing [7] are popularly used for solving the CAP by several authors. In [14], a hybrid meta-heuristic is proposed which is based on a hybridization of a heuristic namely randomized adaptive search procedure ($GRASP$) with a frequency exhaustive strategy. In [1] authors have presented a probabilistic greedy algorithm based on two diversification techniques namely randomization and perturbation. Authors have argued that while deterministic greedy algorithm may get trapped in local optima, probabilistic greedy is sometime helpful. In [18] an elegant technique is presented which first maps a given CAP to a coalesced CAP with reduced search space and the solution of this coalesced CAP is used to solve the original CAP. In [19] a technique is presented where a CAP with non-homogeneous demand is first partitioned into a sequence of smaller subproblems with homogeneous demands only. Solutions of these subproblems then constitute the solution of the original problem.

The existing CAP algorithms have been applied on well-known benchmark problems to evaluate and compare their performance. Most widely used benchmarks are the Philadelphia benchmarks [9, 8, 21, 20, 16, 17, 18]. These benchmarks are defined on a 21-cell network, as shown in Figure 2.1. The demand

Table 2.1: Two different demand vectors for the Philadelphia benchmarks

$Cell$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
D_1	8	25	8	8	8	15	18	52	77	28	13	15	31	15	36	57	28	8	10	13	8
D_2	5	5	5	8	12	25	30	25	30	40	40	45	20	30	25	15	15	30	20	20	25

vectors used for these benchmarks are given by D_1 and D_2 as shown in Table 2.1. The column-i of Table 2.1 refers to the channel demand from cell i. Table 2.2 shows the specifications of these eight benchmark problems (problems 1 through 8) in terms of the specific values of the frequency separation constraints s_0, s_1 and s_2. Here s_0, s_1 and s_2 ($s_0 \geq s_1 \geq s_2$) are the minimum frequency separation required between the calls in the same cell, and in cells at distance one and two apart, respectively. It is assumed that the channel interference does not extend beyond two cells for these benchmark instances. That is, the same channel can be reused to two cells if they are distance three or more apart. The frequency separation matrix $M = (m_{ij})$ can be generated from the values of s_0, s_1 and s_2 as follows: $m_{ii} = s_0$, $m_{ij} = s_1$ if cells i and j are distance one apart, $m_{ij} = s_2$, if they are distance two apart, and $m_{ij} = 0$, if their distance is 3 or more. Apart from these Philadelphia benchmarks, we also consider a practical assignment problem from Helsinki, Finland defined on a 25-node benchmark network [16, 20]. The frequency separation matrix $M = (m_{ij})$ and the demand vector D_3 for this problem (problem 9) are shown in Tables 2.3 and 2.4, respectively. The entry corresponding to the i-th row and j-th column of Table 2.3, i.e., m_{ij}, represents the minimum frequency separation requirement between a call in cell i and a call in cell j ($0 \leq i, j \leq 24$). The column-i of Table 2.4 indicates the channel demand from cell i. We also have considered two relatively larger benchmarks defined on a 55-node network as shown in Fig. 2.2 [16]. For these two benchmarks the values of s_0, s_1 and s_2 are given as 7, 1 and 1, respectively. The demand vectors of these two problems (problems 10 and 11) are given by D_4 and D_5 respectively, as shown in Table 2.5. Some other benchmark instances (problems 12 through 14) are used by [1, 2] and defined on 15-, 30- and 40-cell networks shown in Figures 2.3, 2.4 and 2.5 respectively. All these instances use unit demand per cell. The frequency separation matrix is given by $M = (m_{ij})$ where m_{ij} is 1 when cells i and j are distance one or two apart and 0, otherwise.

Among the Philadelphia benchmarks, problems 1, 3, 4, 5 and 7 are relatively easy as the required span is mainly determined by the value of s_0. As a result, most of the existing algorithms can find optimal solutions for them within few seconds only. The most difficult to solve are the instances - problems 2 and 6 [8, 20]. The lower bounds on the span for problems 2 and 6 are known as 426 and 252 respectively [10, 18, 8]. For problems 2 and 6 many solutions have been proposed previously. For example [9] proposed an algorithm which requires 165 hours to produce a span of 267 for problem 6 in HP Apollo 9000/700 workstation. The genetic algorithm reported in [17] produces optimal solution for both the problems but the computation time was varied between 12-80 hours on a DEC Alpha Workstation. The algorithm reported in [6] combines a sequential heuristic method into a genetic algorithm and produces solutions for problems 2 and 6 with spans 431 and 252 respectively by taking a time of 10 hours. The combined genetic algorithm [8] got optimal solutions for both problems with a running time of 8 and 10 minutes respectively. The frequency exhaustive strategy with rearrangement ($FESR$) proposed in [21] for CAP produces solutions for problems 2 and 6 with spans 432 and 259 respectively. However, the computation time is not mentioned in the paper. An adaptive local-search algorithm presented in [23] produces spans of 432 and 262 for problems 2 and 6 requiring time 200-300 seconds. Randomized saturation degree (RSD) heuristic in combination with a local search (LS) algorithm reported in [20] produces solutions for problems 2 and 6 with spans 426 and 253 respectively with computation time varied between 110-170 seconds. The algorithm in [18] produces optimal solutions for both the problems with computation time varied between 10-20 seconds only. The heuristic in [16] produces the spans of 462 and 272 for problems 2 and 6 with time 9.5 and 7.7 seconds respectively. Most recently the algorithm presented in [19] requires only 5 milliseconds of running time for problems 2 and 6 to result the spans 448 and 267 respectively. From the above discussion, it appears that some algorithms take long computation time but produces results very close to the optimal. Whereas, some others produce results very quickly but the obtained span is far from the optimality. In this report, we have developed an algorithm that produces span close to the optimality and takes time less than 1.77 seconds only even for the most difficult instances. Thus the proposed algorithm is very much suitable for real life application where fast channel assignment is of primary importance while a marginal deviation from optimality may be tolerated.

Table 2.2: Specification of Philadelphia benchmark problems.

$Problems$		1	2	3	4	5	6	7	8
$Frequency$	s_0	5	5	7	7	5	5	7	7
$Separation$	s_1	1	2	1	2	1	2	1	2
$Constraints$	s_2	1	1	1	1	1	1	1	1
$Demand\ vector$		D_1	D_1	D_1	D_1	D_2	D_2	D_2	D_2

Table 2.3: Frequency separation matrix for problem 9.

$$C = \begin{bmatrix}
2 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 2 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 2 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 2 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\
1 & 1 & 1 & 0 & 2 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 2 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 0 & 1 & 2 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 0 & 1 & 1 & 2 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 2 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 2 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\
0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 2 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 2 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 2 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 2 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 2 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 2 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 2 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 2 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 2 & 1 & 1 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 2 & 1 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 2 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 2 & 1 & 1 & 1 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 2 & 1 & 1 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 2 & 1 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 2
\end{bmatrix}$$

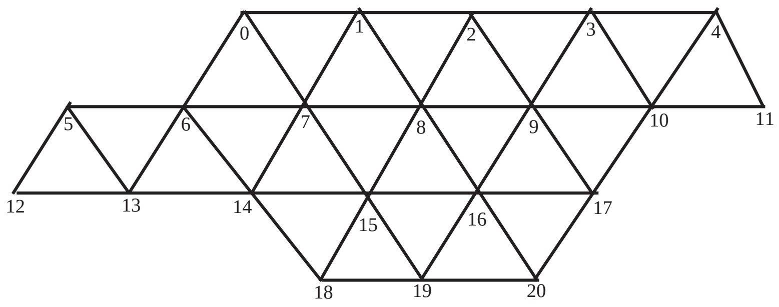

Figure 2.1: The 21-cell benchmark network.

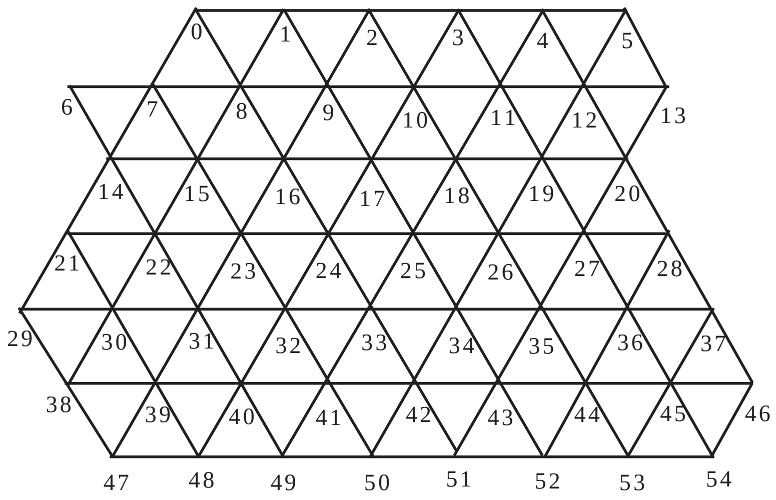

Figure 2.2: The 55-cell benchmark network.

Table 2.4: The demand vector for problem 9.

$Cell$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
D_3	10	11	9	5	9	4	5	7	4	8	8	9	10	7	7	6	4	5	5	7	6	4	5	7	5

Table 2.5: Two demand vectors for 55-node benchmark problems.

$Cell$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
D_4	5	5	5	8	12	25	30	25	30	40	40	45	20	30	25	15	15	30	20	20	25	8	5	5	5	5	5	5
D_5	10	11	9	5	9	4	5	7	4	8	8	9	10	7	7	6	4	5	5	7	6	4	5	7	5	10	11	9

$Cell$	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	
D_4	8	12	25	30	25	30	40	40	45	20	30	25	15	15	30	20	20	25	8	5	5	5	25	8	5	5	5	
D_5	5	9	4	5	7	4	8	8	9	10	7	7	6	4	5	5	7	6	4	5	7	5	6	4	5	7	5	

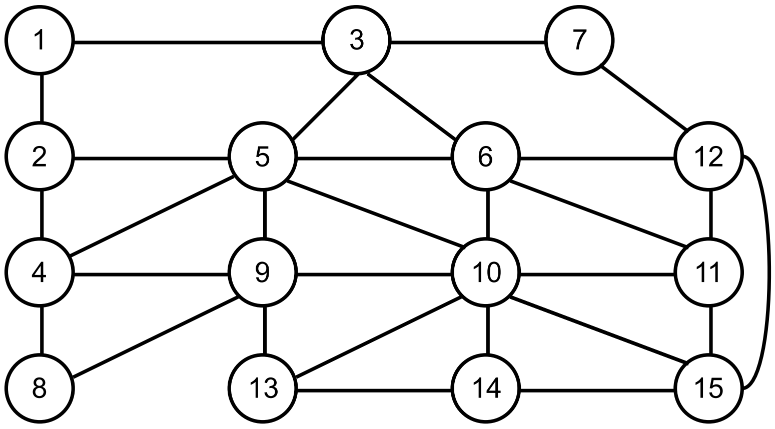

Figure 2.3: The 15 cell benchmark network.

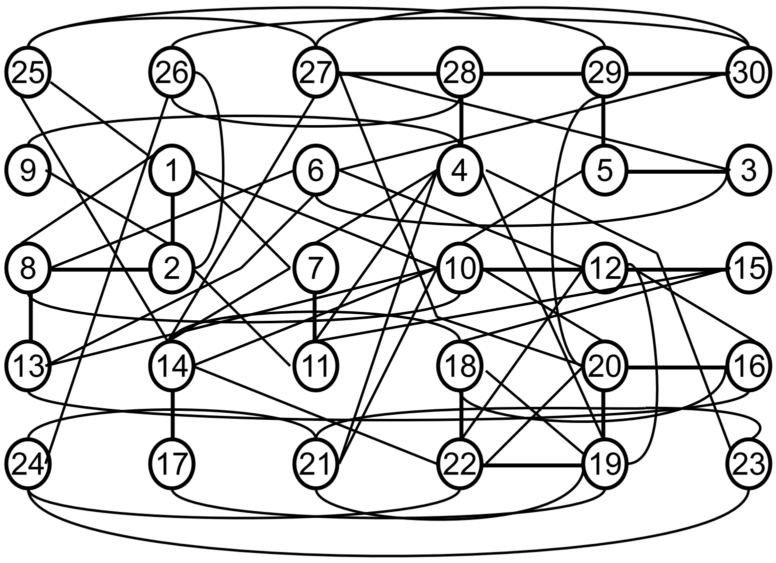

Figure 2.4: The 30 cell benchmark network.

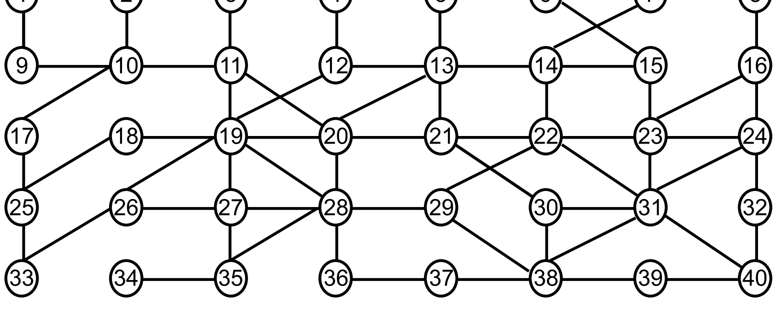

Figure 2.5: The 40 cell benchmark network.

Chapter 3

The CAP graph

The CAP can be represented by a complete edge weighted graph, CAP graph, $G(V, W)$ where each vertex $v \in V$ represents a call and the edge weight w_{uv} of the edge (u, v) represents the minimum frequency separation needed between the calls representing the vertices u and v. We put an edge between two vertices u and v with weight 0 if $w_{uv} = 0$. We can build the CAP graph from the frequency separation matrix and the demand vector. The number of vertices of the CAP graph is equal to the summation of demands across all the cells. We need to find a frequency assignment vector $F = (f_v)$ where f_v represents the frequency assigned to vertex v. Our goal is to find $f_v \ \forall v \in V$, such that $\forall u, v \in V$, $|f_u - f_v| \geq w_{uv}$ and $max(F) = \max_{v \in V} f_v$ is minimized. Under this CAP graph representation, the CAP is basically a graph coloring problem where each vertex has to be assigned only one color instead of multiple colors required as per the traditional representation of the problem. We can now state the following simple result.

Lemma 1. *Let g be the gcd of the positive edge weights of the CAP graph $G(V, W)$. If $g > 1$ then we can build a new CAP graph $G'(V, W')$ where $W' = W/g$. The frequency assignment vector F for the original CAP graph $G(V, W)$ can be deduced from the frequency assignment vector F' of $G'(V, W')$ by $F = gF'$.*

In view of the result above, from now onwards we will assume that gcd of the positive edge weights of the CAP graph is 1. The following example demonstrate the construction of the CAP graph from the demand vector and frequency separation matrix.

Example 1. *Consider a CAP represented by (X, M, Λ) where $X = \{0, 1, 2\}$ and $\Lambda = (2, 1, 1)$. The frequency separation matrix M for this example is given by:*

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

Figure 3.1 shows the corresponding CAP graph where 4 vertices represent the 4 calls. Vertices a_0 and a_1 represent the two calls generated at cell 0, vertex b_0 represents the call generated at cell 1 and vertex c_0 represents the call generated at cell 2. The edges of the CAP graph are labeled with weights according to matrix M.

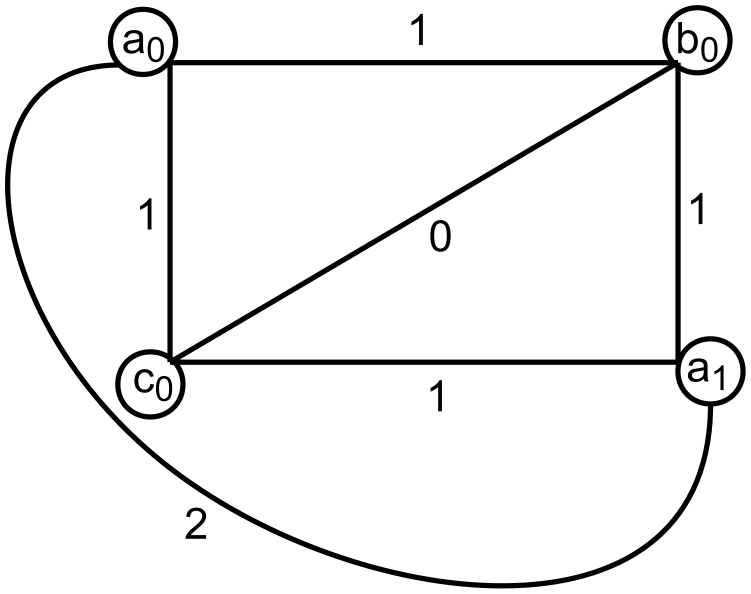

Figure 3.1: An example CAP graph

Chapter 4

The proposed algorithm

The proposed algorithm is based on the CAP graph representation of the problem and we have used the frequency exhaustive (FE) strategy in our approach. In FE, we start with an ordering of vertices and then color vertices one by one following that order. Let $(v_0, v_1, \cdots, v_{n-1})$ be an ordering of n vertices of the CAP graph. In FE, we apply color 0 to v_0 and then for each v_i we check the frequency separation constraints with the already allocated vertices $v_0, v_1, \cdots, v_{i-1}$ and put the minimum color that satisfies these constraints. We will introduce the *probabilistic ordering*, a *forced assignment* phase and a *compression* phase to reduce the span while coloring the CAP graph. Finally a *smoothing* phase is applied to increase the channel utilization. We break the boundary of determinism and define a *probabilistic ordering* of the vertices of the CAP graph. Then we will start coloring vertices using FE following that probabilistic ordering, but when color of a vertex will exceed the maximum color of previously allocated vertices we will go for a *forced assignment* phase to reduce the so far obtained span. After all vertices will be colored we will use a *compression* phase where we choose the maximum colored vertex and try to reduce its color by increasing some of the other vertices' colors. We iteratively go on this compression phase till no further reduction is possible. The *smoothing* phase is applied over the assignment obtained by the compression phase. It will consider the vertices in the derived probabilistic ordering and try to reduce the color of each vertex by putting the minimum color that satisfies all the already allocated vertices. This phase will be iteratively applied as long as a vertex's color can be reduced. Formally these four phases are described below.

4.1 Probabilistic ordering

In probabilistic ordering, we choose the vertex to be present in the beginning of an order with a probability. Let $d_{v_i} = \sum_{x \in V} w_{v_i x}$ be the degree of vertex v_i. Note that the degree of a vertex is nothing but the sum of the edge weights of the edges incident to that vertex. We give the high degree vertex a higher probability to be at the beginning of an order. Let the *degree vector* corresponding to the vertices $v_0, v_1, \cdots, v_{n-1}$ is $D = (d_{v_0}, d_{v_1}, \cdots, d_{v_{n-1}})$, where d_{v_i} is the degree of vertex v_i. We now define $\omega_{v_x} = (d_{v_x} + r_{v_x})\ mod\ D_{max}$ where D_{max} is the maximum value in the degree vector D and r_{v_x} is a random number belonging to $\{0, 1, \cdots, (D_{max} - 1)\}$. Then we sort the vertices according to the decreasing order of their ω values. Essentially this step produces an order where high degree vertex has higher probability of being at the beginning.

4.2 Forced assignment

In FE if a vertex's color exceeds the maximum of the previously allocated vertices' colors, there is no chance to look back and change the color of any already allocated vertex in order to reduce the span. In the forced assignment we tries to look back to the already allocated vertices and find a way to reduce the so far obtained span. Basically, it tries to change one of the previously allocated vertex's color to reduce the span, if possible. While coloring a CAP graph with FE if $f_{v_i} > max(f_{v_0}, f_{v_1}, \cdots, f_{v_{i-1}})$ then we will go for a forced assignment phase. The strategy is formally presented in Algorithm 1 and demonstrated in Example 2.

Example 2. *Consider the CAP graph shown in Figure 4.1 (a). The label associated with an edge represents the weight of that edge. The edges with weight zero are not shown in the figure. We start FE with order* (e, c, a, d, b). *We will put 0 at e, 2 at c, and 4 at a. Next we go for forced assignment as color of a exceeds the previous maximum. In the first iteration, we will first free the colors of e and a and then put the minimum color at a that satisfied the frequency separation constraint with the already allocated vertex c. Next we put the minimum color at e that satisfied the constraint with c and a. Thus a will get the color 0 and e will get 4. So forced assignment could not reduce the span in this iteration. In the next iteration, we will free the colors of c and a and then put the minimum color at a that satisfied the constraint e. Next we put the minimum color at c that satisfied the constraints with e and a. Thus a will get the color 1 and c will get 3. Thus in this iteration the span is reduced to 3 from 4. Up to this the coloring of* (e, c, a) *is* $(0, 3, 1)$. *If we continue with FE, we will allocate 3 at d and 5 at b. Once again we*

Algorithm 1: *Forced Assignment*

```
   Input: $v_i$, $F = (f_{v_j})$
   Output: $F$
1  $F_{max} = f_{v_i}$;
2  for $\forall v_x \in \{v_0, v_1, \cdots, v_{i-1}\}$ do
3      if $w_{v_x v_i} > 0$ then
4          $b_i = f_{v_i}$, $b_x = f_{v_x}$;
5          Erase the colors of $v_i$ and $v_x$;
6          Set $f_{v_i}$ = minimum color that satisfies the constraints with all already
           allocated vertices $\{v_0, v_1, \cdots, v_{x-1}, v_{x+1}, \cdots, v_{i-1}\}$;
7          Set $f_{v_x}$ = minimum color that satisfies the constraints with all already
           allocated vertices $\{v_0, v_1, \cdots, v_{x-1}, v_{x+1}, \cdots, v_i\}$;
8          if $max(f_{v_0}, f_{v_1}, \cdots, f_{v_i}) < F_{max}$ then
9              Return $F$;
10         $f_{v_i} = b_i$, $f_{v_x} = b_x$;
11 Return $F$;
```

will go for forced assignment but now we will see that this phase cannot further reduce the span. So, we end up with an assignment $(0, 3, 1, 3, 5)$ *which has been shown in the figure. The label [x] associated with a vertex represent that color x has been assigned to that vertex.*

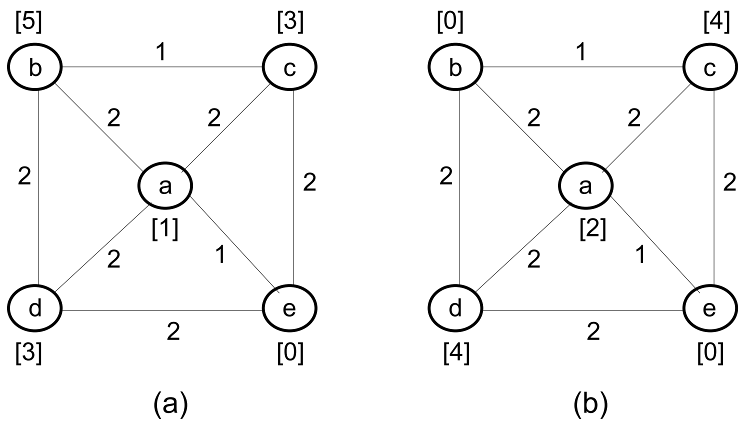

Figure 4.1: Example for (a) forced assignment (b) compression.

4.3 Compression

The FE with forced assignment will produce a conflict-free assignment of the CAP graph. We now apply compression over this conflict-free assignment to further reduce the span, if possible. We choose the maximum colored vertex and then reduce the span by reducing the color of this vertex by increasing the colors of other vertices. Note that this strategy is independent of the frequency assignment we had started with. The only criteria is that it should be conflict-free. In this compression phase, we will first compress a triangle (a complete graph of 3 vertices) and then propagate the effect of the *triangle compression* over the whole graph. To formally describe the compression phase we need to consider the following results.

Theorem 1. *If (f_u, f_v, f_x) is a conflict-free frequency assignment of a triangle $T = (u, v, x)$, where $f_u \leq f_x \leq f_v$, then $(f_u^{'}, f_v^{'}, f_x^{'}) = (f_u, f_u^{'}+w_{uv}, max(f_x, f_v^{'} + w_{vx}))$ is also a conflict-free frequency assignment of T.*

Proof. In order to show that $(f_u^{'}, f_v^{'}, f_x^{'})$ is a conflict-free frequency assignment of $T = (u, v, x)$, we have to show that this assignment satisfies the frequency separation constraints. It is obvious that $|f_v^{'} - f_u^{'}| = w_{uv}$ since $f_v^{'} = f_u^{'} + w_{uv}$. Since $f_x^{'} = max(f_x, f_v^{'}+w_{vx})$, depending on the relative values of f_x and $f_v^{'}+w_{vx}$ we have the following two cases.

When $f_x \geq f_v^{'}+w_{vx}$, $f_x^{'} = f_x$. Hence $|f_x^{'}-f_v^{'}| \geq w_{vx}$. Also $|f_x^{'}-f_u^{'}| = |f_x - f_u| \geq w_{ux}$ since (f_u, f_v, f_x) is given to be a conflict-free frequency assignment of T and $f_u^{'} = f_u$.

When $f_v^{'} + w_{vx} \geq f_x$, $f_x^{'} = f_v^{'} + w_{vx} \geq f_x$. This implies $|f_x^{'} - f_v^{'}| = w_{vx}$. Now, $|f_x^{'} - f_u^{'}| \geq |f_x - f_u^{'}| = |f_x - f_u| \geq w_{ux}$, since $f_u^{'} = f_u$ and (f_u, f_v, f_x) is given to be a conflict-free frequency assignment of T. Hence $|f_x^{'} - f_u^{'}| \geq w_{ux}$. Hence the result. □

Corollary 1. *The largest frequency in the assignment $(f_u^{'}, f_v^{'}, f_x^{'})$ of T is $f_x^{'}$.*

Proof. Since $f_v^{'} = f_u^{'} + w_{uv}$, $f_u^{'} \leq f_v^{'}$. Since $f_x^{'} = max(f_x, f_v^{'} + w_{vx})$, depending on the relative values of f_x and $f_v^{'} + w_{vx}$ we have the following two cases. When $f_v^{'} + w_{vx} \geq f_x$, $f_x^{'} = f_v^{'} + w_{vx}$. This implies $f_x^{'} \geq f_v^{'}$. For the other case when $f_x \geq f_v^{'} + w_{vx}$, $f_x^{'} = f_x$. So $f_x^{'} \geq f_v^{'}$. Hence $f_u^{'} \leq f_v^{'} \leq f_x^{'}$. □

Notice that f_v and $f_x^{'}$ are the maximum in the previous and next assignment of T respectively. Now if we found that $f_x^{'} < f_v$ then this reassignment certainly minimize the maximum frequency of the previous assignment of T. We will call this new assignment as $triangle\ compression$ of T.

Corollary 2. *In the assignment* $(f_u^{'}, f_v^{'}, f_x^{'})$ *of* T*,* $f_u^{'} - f_u = 0$*,* $f_v^{'} - f_v \leq 0$ *and* $f_x^{'} - f_x \geq 0$*. That is, only the new frequency assigned to* x *is increased from its previous value and frequency of* u *remains unchanged.*

Proof. We know that $f_u^{'} = f_u$ and $f_x^{'} = max(f_x, f_v^{'} + w_{vx})$. So $f_x^{'} \geq f_x$. We are left to show $f_v^{'} \leq f_v$. Now $|f_v - f_u| \geq w_{uv}$ since (f_u, f_v, f_x) is given to be a conflict-free assignment of T. Since we know that $f_v \geq f_u$, $f_v \geq f_u + w_{uv}$. Since $f_v^{'} = f_u + w_{uv}$, $f_v^{'} - f_v \leq 0$. □

We will call $(f_x^{'} - f_x)$ as $extra$ value of the new assignment because only frequency of vertex x could increase and the $extra$ contain that amount of increment. Also we denote u as the μ vertex with respect to the new assignment as its frequency will remain the minimum in the triangle. In our strategy (μ , $extra$) pair will be used to spread the effect of triangle compression over the whole graph. To compress the overall span using the triangle compression, we will follow the strategy described in Algorithm 2. We will give input as the maximum colored vertex v_i, the frequency assignment vector F and get a reassignment F'' as output.

Algorithm 2: *Compression*

```
Input: $v_i$, $F$
Output: $F''$
1  $F'' = F$;
2  for $\forall$ triangle $T \in G$ such that $T$ has a triangle compression do
3      Set $F' = F$;
4      for $\forall x \in T$ do
5          Set $f_x^{'}$ according to triangle compression;
6      for $\forall x \in (G \setminus T)$ do
7          if $f_x > f_\mu$ then
8              Set $f_x^{'} = f_x + extra$;
9      if $F'$ is conflict-free and $\max_x(f_x^{'}) < \max_x(f_x^{''})$ then
10         $F'' = F'$;
11 return $F''$;
```

The assignment F' obtained by executing Lines 3 to 8 in Algorithm 2 may or may not be conflict-free. We now state the following theorem to know the number of comparison required to test whether F' is conflict-free or not.

Theorem 2. *To ensure whether* $F^{'}$ *obtained in Algorithm 2, is conflict-free or not we need to check only the constraints between* v_i *and vertices of* $(G \setminus \{v_i\})$*.*

Proof. To prove the above statement first we will partition vertices of $(G \setminus \{v_i\})$ into 2 disjoint sets H and L. Here H contains all those vertices x for which $f_x^{'} > f_\mu$, and L contains the remaining vertices from $(G \setminus \{v_i\})$. Observe from Algorithm 2 that frequencies assigned to the vertices in H have been increased by $extra$ and frequencies of vertices of L remained unchanged. So there cannot be any conflict in between vertices of H as their relative frequency difference remains same. Same logic applies for the vertices of L. Also there will be no conflict in between vertices of H and L because their frequency difference could only increase. Thus to ensure whether $F^{'}$ is conflict-free, it is sufficient to check the constraints between v_i and $(G \setminus \{v_i\})$. Thus it requires only $O(n)$ comparison to check whether $F^{'}$ is conflict-free. □

The F'' returned by Algorithm 2 either contains the previous assignment F or a conflict-free assignment F' with a reduced span. If it contains an assignment with reduced span we will call this reassignment as *compression*. Example 3 illustrates this compression phase.

Example 3. *Consider the CAP graph and its assignment* $(0, 3, 1, 3, 5)$ *corresponding to the order* (e, c, a, d, b) *as produced by the FE with forced assignment phase shown in Figure 4.1 (a). Now we will try to compress the span further by our compression phase. Notice that* b *has the largest color assigned. Thus we will consider all the* $\binom{4}{2}$ *triangles that includes* b *as a vertex together with their triangle compressions. Among them if we consider triangle* (e, b, a) *and its triangle compression we will see that by increasing the color of* a *by* 1 *we can reduce the color of* b *to* 0*. The extra value of this triangle compression is* 1 *and* e *becomes the* μ *vertex and hence its color remain unchanged. Among the colors of the set of vertices* $\{e, c, a, d, b\} \setminus \{e, b, a\} = \{c, d\}$ *whose colors are more than 0 will be increased by 1. Thus after compression the color assignment becomes* $(0, 4, 2, 4, 0)$ *as shown in Figure 4.1 (b) which is a conflict-free assignment. Thus total span is reduced by* 1.

4.4 Smoothing

After the compression phase we get an assignment of the vertices of graph G. In this assignment, some of the vertices's color may be expanded due to the addition of $extra$ value in the compression phase. If we keep the assignment in this form the potential to satisfy further demands may be reduced. Thus we will try to reduce the colors of some of the vertices so as to cope up with the demand fluctuations that may arise in a latter phase. To take into consideration this effect, we introduce a smoothing phase whose objective is to reduce the colors of some of the vertices while satisfying the required frequency separation constraints. To do this, we introduce a metric called channel utilization as $Y = \sum_{v \in V} f_v/|V|$. Note that Y is nothing but the summation of all colors by total number of calls. If Y is high then we can say that more higher colors are in F. In contrast to this, where Y is less we can say that more lower colors are there in F. Our objective is to reduce Y.

In the smoothing phase we will start with an order of vertices and then traverse through vertex by vertex in that order. While visiting a vertex we will first free its color and then find the minimum color that satisfies constraints with all already allocated vertices. Once every vertices are visited we will check whether the value of Y is reduced. If yes, we will repeat this process as long as a reduction is possible and then terminate. The algorithm is formally described in Algorithm 3.

Algorithm 3: *Smoothing*

```
   Input: $V$, $F$
   Output: $F, Y$
1  $Y = \sum_{v \in V} f_v$;
2  for $\forall v \in V$ do
3      Erase color of $v$;
4      $f_v$ = minimum color that satisfies constraints with all already allocated
       vertices;
5  $Y_{new} = \sum_{v \in V} f_v$;
6  while $Y > Y_{new}$ do
7      $Y = Y_{new}$;
8      for $\forall v \in V$ do
9          Erase color of $v$;
10         $f_v$ = minimum color that satisfies constraints with all already allocated
           vertices;
11     $Y_{new} = \sum_{v \in V} f_v$;
12 $Y = Y/|V|$;
13 return $F, Y$;
```

4.5 The overall algorithm

We first find the probabilistic ordering $(v_0, v_1, \cdots, v_{n-1})$ of the vertices of the CAP graph. Then we assign color 0 to v_0 and then following this order for each vertex we put the minimum color that satisfies the constraints with already allocated vertices. But in this process when color of a vertex exceeds the previous maximum color we will go for a forced assignment phase. Finally when all vertices are colored we will go for a compression phase, where we will try to compress the span till it is possible. Finally the smoothing phase is applied to increase the channel utilization. Formally the overall algorithm is presented in Algorithm 4 where set of vertices V and the degree vector D are the inputs and final assignment F' is the output.

Algorithm 4: *Proposed Algorithm*

```
   Input: $V, D$
   Output: $F'$
1  for $\forall v \in V$ do
2      $\omega_v = (d_v + r_v)\ mod\ max(D)$;
3  Sort vertices according to decreasing order of their $\omega$ values;
4  Let $(v_0, v_1, \cdots, v_{n-1})$ be this sorted order;
5  Set $F = \phi$;
6  for $\forall v_i \in V$ do
7      $f_{v_i}$ = minimum color that satisfies the constraints with already allocated
       vertices;
8      if $f_{v_i} > max(f_{v_0}, f_{v_1}, \cdots, f_{v_{n-1}})$ then
9          $F = ForcedAssignment(v_i, F)$;
10 Let $v_x$ be the maximum colored vertex of $F$;
11 $F' = Compression(v_x, F)$;
12 while $\max\limits_y f_y^{'} < \max\limits_y f_y$ do
13     $F = F'$;
14     Let $v_x$ is the maximum colored vertex of $F$;
15     $F' = Compression(v_x, F)$;
16 Smoothing($V$,$F'$);
17 return $F'$;
```

Chapter 5

Generalized algorithm

In the generalization of the algorithm we had generalized the forced assignment and compression phase. In forced assignment we picked up a tuple instead of picking a pair and in compression we generalized our triangle compression for higher clique. More formally the steps are explain bellow.

5.1 Generalized forced assignment

The force assignment phase is called when we have an conflict free assignment of the vertices $v_0, v_1, \cdots v_i$ with $f_{v_i} > \max_{k=0}^{i-1} f_{v_k}$. Now we form i pairs of vertices by selecting one from $v_0, v_1, \cdots v_{i-1}$ and v_i. For a pair (v_i, v_k) we first free the colors of v_i and v_k and then allocate minimum color that satisfies the constraints with all already allocated vertices to v_i and v_k in this order. Now if this operation reduces the span we terminate. Else we return back the original colors to those vertices and go for the next pair.

In the generalization, we choose a *k-tuple* instead of a pair. We will make all possible k-tuples (where k is constant) where v_i is a member of that tuple. Let such a k-tuple be $(v_i, x_1, x_2, \cdots, x_{k-1})$. Now we color them in this order. While coloring we always put the minimum color that satisfies constraints with all the already allocated vertices. During the process, once we get reduced span, we will terminate. Else we return back the previous color to those vertices and go for the next tuple. More formally the algorithm is described in Algorithm 5.

5.2 Generalized compression

We start with a compression algorithm which first compress a triangle then propagate its effect throughout the graph and do overall compression. But the traingle

Algorithm 5: *Generalize Forced Assignment*

```
   Input: $v_i$,$k$, $F$
   Output: $F$
1  Set $maxCol = \max_{l=0}^{i}(f_{v_l})$;
2  for All possible combination of $k-1$ vertices $x_1, x_2, \cdots, x_{k-1}$ from
   $\{v_0, v_1, \cdots v_{i-1}\}$ do
3      Choose an arbitrary permutation $(x_1, x_2, \cdots, x_{k-1})$ of the selected vertices;
4      First save and then free the colors of $x_1, x_2, \cdots, x_{k-1}$ and $v_i$;
5      Allocate minimum color that satisfies the already allocated vertices to the
       vertices $x_1, x_2, \cdots, x_{k-1}, v_i$ in order;
6      Let $f_{v_k}$ be the color assigned to $v_k$;
7      if $\max_{k=0}^{i}(f_{v_k}) < maxCol$ then
8          return $F$;
9      Return back the saved values to the vertices $x_1, x_2, \cdots, x_{k-1}, v_i$;
10 return $F$;
```

compression could be generalized further. It could be generalized for a k-clique instead of a triangle. The generalization is explained in following.

Theorem 3. *If $(v, u, x_1, x_2, ...x_{k-2})$ is a k-clique and $f_v, f_u, f_{x_1}, f_{x_2} \cdots f_{x_{k-2}}$ is a conflict free assignment of it, where $f_u \leq f_{x_1} \leq f_{x_2} \cdots \leq f_{x_{k-2}} \leq f_v$ then $(f_v^{'}, f_u^{'}, f_{x_1}^{'}, f_{x_2}^{'} \cdots, f_{x_{k-2}}^{'})$ is also a conflict free assignment where $f_u^{'} = f_u$, $f_v^{'} = f_u^{'} + w_{uv}$ and $\forall i,\ f_{x_i}^{'} = \max(f_{x_i}, f_v^{'} + w_{vx_i}, \max_{j=1}^{i-1}(f_{x_j}^{'} + w_{x_j x_i}))$.*

Proof. In the above reassignment color of u will remain the same. Since $f_v^{'} = f_u^{'} + w_{uv}$, color of v is conflict free with u. For any i, $f_{x_i}^{'} \geq f_{x_i}$ and $f_{x_i}^{'} \geq f_v^{'} + w_{vx_i}$ and $\forall j \in [1 \cdots i-1]$, $f_{x_i}^{'} \geq f_{x_j}^{'}$ because $f_{x_i}^{'} = \max(f_{x_i}, f_v^{'} + w_{vx_i}, \max_{j=1}^{i-1}(f_{x_j}^{'} + w_{x_j x_i}))$. Thus $|f_{x_i}^{'} - f_u^{'}| \geq |f_{x_i} - f_u| \geq w_{ux_i}$ as f_{x_i} and f_u were conflict free. Also $|f_{x_i}^{'} - f_v^{'}| \geq w_{vx_i}$ as $f_{x_i}^{'} \geq f_v^{'} + w_{vx_i}$. Also $\forall j \in [1 \cdots i-1] |f_{x_i}^{'} - f_{x_j}^{'}| \geq w_{x_i x_j}$ as $\forall j \in [1 \cdots i-1] f_{x_i}^{'} \geq f_{x_j}^{'}$. Hence the proof. □

Corollary 3. *In the above reassignment, $f_u^{'} \leq f_v^{'} \leq f_{x_1} \leq f_{x_2} \cdots \leq f_{x_{k-2}}$.*

Proof. Since $f_v^{'} = f_u^{'} + w_{uv}$, we get $f_u^{'} \leq f_v^{'}$. Since $f_{x_i}^{'} = \max(f_{x_i}, f_v^{'} + w_{vx_i}, \max_{j=1}^{i-1}(f_{x_j}^{'} + w_{x_j x_i}))$, $f_{x_i}^{'} \geq f_v^{'} + w_{vx_i} \geq f_v^{'}\ \forall i$ and $f_{x_i}^{'} \geq f_{x_j}^{'} + w_{x_i x_j} \forall i\ \&\ \forall j < i$. This implies $f_{x_i}^{'} \geq f_{x_j}^{'} \forall i\ \&\ \forall j < i$. Hence the proof. □

In the above reassignment, the color of u remain unchained, color of v decreases and color of x_i $\forall i$ may increase or remain the same. We will call the above reassignment as *k-clique compression* and u as μ vertex and the value $\max_{i=1}^{k-2}(f'_{x_i} - f_{x_i})$ as $extra$.

Algorithm 6: *Generalize Compression*

```
   Input: $v_i$,$k$,$F$
   Output: $F''$
1  $F'' = F$;
2  for $\forall$ k-clique $K_k \in G$ such that $K_k$ has a k-clique compression do
3      Set $F' = F$;
4      for $\forall x \in K_k$ do
5          Set $f'_x$ according to k-clique compression;
6      for $\forall x \in (G \setminus K_k)$ do
7          if $f_x > f_\mu$ then
8              Set $f'_x = f_x + extra$;
9      if $F'$ is conflict-free and $\max_x(f'_x) < \max_x(f''_x)$ then
10         $F'' = F'$;
11 return $F''$;
```

Theorem 4. *To ensure whether F' in Algorithm 6 is conflict free we have to check the constrains between $S = \{v_i, x_1, x_2 \cdots x_{k-2}\}$ and vertices of $G \setminus S$ and within the vertices of S.*

Proof. We divide the set of vertices in G into 3 subsets S, L and H. Let $S = \{v_i, x_1, x_2 \cdots x_{k-2}\}$, $L = \{\forall x \in V f'_x \leq f_\mu\}$ and H be the set of remaining vertices. Note that any two vertices in L are conflict free as their colors remain unchanged. Any two vertices of H are also conflict free as their colors are increased by a constant value. So we have to check the constraints between vertices of $G \setminus S$ and S and within the vertices of S. □

5.3 The overall generalized algorithm

The overall algorithm is same as Algorithm 4 except we use here the generalized forced assignment and compression phases. We pass another constant k which will determine the size of clique we are taking. In formal words the overall generalized algorithm is presented in Algorithm 7.

Algorithm 7: *Proposed Generalized Algorithm*

```
   Input: $V$, $D$,$k$
   Output: $F'$
1  for $\forall v \in V$ do
2      $\omega_v = (d_v + r_v)\ mod\ max(D)$;
3  Sort vertices according to decreasing order of their $\omega$ values;
4  Let $(v_0, v_1, \cdots, v_{n-1})$ be this sorted order;
5  Set $F = \phi$;
6  for $\forall v_i \in V$ do
7      $f_{v_i}$ = minimum color that satisfies the constraints with already allocated
       vertices;
8      if $f_{v_i} > max(f_{v_0}, f_{v_1}, \cdots, f_{v_{n-1}})$ then
9          $F = GeneralizedForcedAssignment(v_i, k, F)$;
10 Let $v_x$ be the maximum colored vertex of $F$;
11 $F' = GeneralizeCompression(v_x, k, F)$;
12 while $\max_y f_y^{'} < \max_y f_y$ do
13     $F = F'$;
14     Let $v_x$ is the maximum colored vertex of $F$;
15     $F' = GeneralizeCompression(v_x, k, F)$;
16 $Smoothing(V, F')$;
17 return $F'$;
```

Chapter 6

Complexity of the algorithm

Finding degree of a vertex takes $O(n)$ time. Thus for all vertices it takes $O(n^2)$ time. Sorting the vertices based on these computed values takes $O(n \log n)$ time. So, overall complexity of finding a probabilistic ordering is $O(n^2)$. To compute the complexity of frequency exhaustive with forced assignment phase, we need to know the worse case span that may result by this phase. The following theorem computes this worse case span.

Theorem 5. *Let* $\Delta = \sum\limits_{u,v} w'_{uv}$ *where* $w^{'}_{uv} = 1$ *if* $w_{uv} \geq 1$*, and* 0*, otherwise. The upper bound of the span obtained by frequency exhaustive with forced assignment applied on the* CAP *graph* $G(V, W)$ *is* $\max\limits_{uv}(w_{uv}).\Delta$*.*

Proof. We first transfer the CAP graph $G(V, W)$ to $G'(V, W')$ where $w^{'}_{uv}$ = 1 if $w_{uv} \geq 1$, and 0, otherwise. Also Δ is the maximum degree of a vertex in G'. We know that for any graph a upper bound of span obtained by frequency exhaustive algorithm is Δ. For forced assignment we only put the minimum color that satisfied constraints with already allocated vertices. Since a color can conflict with at most Δ colors in G', FE with forced assignment cannot produce a span greater than Δ. Now we replace all the edge weight 1 of G' by $\max\limits_{uv}(w_{uv})$ and make a graph G''. Now, according to Lemma 1, $\max\limits_{uv}(w_{uv}).\Delta$ will be the worst case span of the G''. It is evident that any conflict-free assignment of G'' is also a conflict-free assignment of G. Thus $\max\limits_{uv}(w_{uv}).\Delta$ is a trivial upper bound of the span of G. $\square$

Note that in FE we visit each node only once and assign the minimum color that satisfied the constrains with already allocated vertices. Thus finding the minimum color for a vertex takes $O(nk)$ where $k = \max\limits_{uv}(w_{uv}).\Delta$, in the worse case.

If color of v_i exceeds the previous maximum color, we will go for a forced assignment phase. In forced assignment, we consider $O(n)$ pairs where v_i is a member of that pair and for each such pair, we need $O(nk)$ time. So to apply forced assignment on v_i it takes $O(n^2k)$ time. In the worse case, we may have to apply forced assignment for $O(n)$ vertices, and thus the total time complexity of this phase becomes $O(n^3k)$.

We now compute the complexity of the compression phase. In this phase, we make $O(n^2)$ triangle and for each triangle we perform triangle compression and then propagate its effect to the whole graph. This propagation takes $O(n)$ time for each triangle. For each triangle, to check whether the obtained reassignment is conflict-free or not it takes $O(n)$ time as stated in Theorem 2. Thus one execution of the compression phase takes $O(n^3)$ time. We iteratively apply compression till no improvement in span is obtained. At each iteration the span will be reduced by at least one or should remain unchanged. If the span remain unchanged, the algorithm terminates. So the total number of iteration could be at most $k = \max\limits_{uv} w_{uv}\Delta$. So the total complexity of this phase becomes $O(n^3k)$.

We now compute the complexity of the smoothing phase. In this phase, we erase the color of a vertex and then find the minimum color that satisfies all the already colored vertices. This step requires $O(nk)$ time for each vertex. Thus for all vertices it requires $O(n^2k)$. This step will be executed as long as a reduction of at least one color is possible. Thus in the worse case time complexity of this phase becomes $O(n^2k^2)$. So the total complexity of the proposed algorithm is $O(n^3k + n^2k^2)$.

We now compute the complexity of the generalized algorithm. The complexity of the probabilistic ordering and the smoothing phase remain same. The generalized forced assignment chooses $(\rho - 1)$-tuple so its complexity will become $O(n^\rho k)$. The generalized algorithm considers ρ-clique compression which requires $O(n^\rho)$ complexity. Thus the total complexity for the generalized algorithm becomes $O(n^\rho k + n^2k^2)$.

Chapter 7

Simulation results

7.1 Experiment on graphs with edge weight $\in \{0, 1\}$

We first simulate our proposed algorithm on graphs with edge weights $\in \{0, 1\}$. For this type of graph, we have the following result.

Theorem 6. *If a CAP graph have only edge weights $\in \{0, 1\}$ and colored by FE with forced assignment then the compression phase cannot further reduce its span.*

Proof. Suppose a CAP graph G has edge weights $\in \{0, 1\}$ and vertex v has the largest color obtained after coloring G by applying FE with the forced assignment. Assume that v has color k. As this is a graph with edge weights $\in \{0, 1\}$, v must have neighbors with color $0, 1, \cdots, k-1$. In any triangle compression, the $extra$ value should be at least 1. This will increase the color of the vertex which is assigned with color $k-1$ and thus the overall span remain the same as k. In other case, if the vertex with color $k-1$ is the member of that triangle, then $k-1$ will be the 2nd largest color of that triangle. In that case too, its color will be increased by at least 1 and hence the span will remain as k. □

However, the FE and forced assignment will work fine on this kind of graph. To judge quality of the results obtained by our proposed greedy, we generate random graph $G(n, p)$ with n vertices and there is an edge between two vertices with probability p, to estimate the chromatic no $\chi(n, p)$. For each n we generate 10 graphs and for each graph we consider 10 ordering and the average span is reported. We know that the upper bound of $E[\chi(n, p)] \sim \frac{nlog(\frac{1}{1-p})}{log(n)}$ [12]. In Fig 7.1, we compare the obtained span by our proposed algorithm with this theoretical upper bound. The results show the same trend of the spans but a constant factor deviation from the upper bound. For $p = 0.5$, we plot the span for $n = 1$ to 500 in

Fig 7.1. Here X-axis contains the values of n and Y-axis is the value of $\chi(n, 0.5)$. The smooth curve represents the upper bound and the rough curve represents the obtained $\chi(n, 0.5)$. The results for $p = 1/3$ and $p = 2/3$ have been shown Figs. 7.3 and 7.2 respectively.

In Table 7.1, the first column shows the value of $X(150, 0.5)$ and the second column shows the percent of occurrence of that value. As shown in the table the mean span obtained is 24.73. From this table, we can say that the values are highly concentrated around the expected value. This statement satisfies the expression given by [11] that is $P(|\chi(n,p) - E[\chi(n,p)]| > \epsilon) < 2exp(\frac{\epsilon^2}{2n})$. Also notice only 2 values have high percentage, rest are so small. This is also following a result of [13] for random graphs. Table 7.2 shows the results for $p = 0.33$ where the mean obtained is 17.13. Also Table 7.3 shows the results for $p = 0.67$ where the mean obtained is 34.14. Experimentally, we get $|\chi(n,p) - E[\chi(n,p)]| \in [0, 4]$ by considering n up to the value of 500.

Table 7.1: Histogram of 150 vertex graph with $p = 0.5$ where mean span is 24.73

$\chi(150, 0.5)$	$Percent$
22	0.067
23	4.44
24	33.37
25	47.47
26	13.71
27	0.911
28	0.031

Table 7.2: Histogram of 150 vertex graph with $p = 0.33$ where mean is 17.13

$\chi(150, 0.33)$	$Percent$
15	0.1956
16	14.2844
17	58.8267
18	25.3956
19	1.2889
20	0.0089

Table 7.3: Histogram of 150 vertex graph with $p = 0.67$ where mean is 34.14

$\chi(150, 0.33)$	$Percent$
31	0.1467
32	3.2978
33	20.3200
34	41.7867
35	27.8889
36	6.0844
37	0.4667
38	0.0089

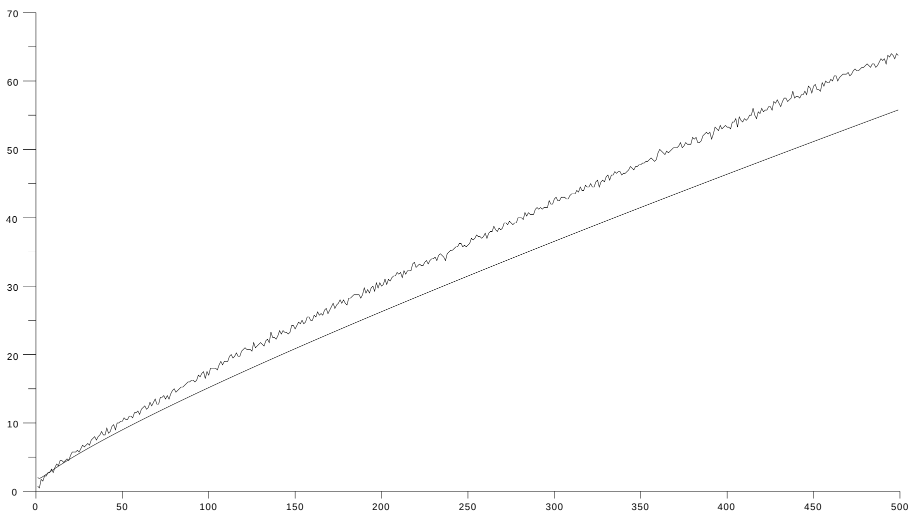

Figure 7.1: n vs $E[\chi(n, 0.5)]$

7.2 Experiment on random weighted graph

To simulate the behavior of our generalized algorithm, we have generated few random graphs and applied our algorithm for k-clique where $k = 3, 4, 5$. We generate random graphs on 50, 100 and 200 vertices. For each graph, we choose edge weights randomly from $[0 \cdots 3]$. We have generated 100 such graphs for each case. We simulate our algorithm for each graph 10 times and report both the best and average span we obtained. Finally we take the mean of those best and average values by considering all 100 graphs. In Table 7.4 best results' mean is

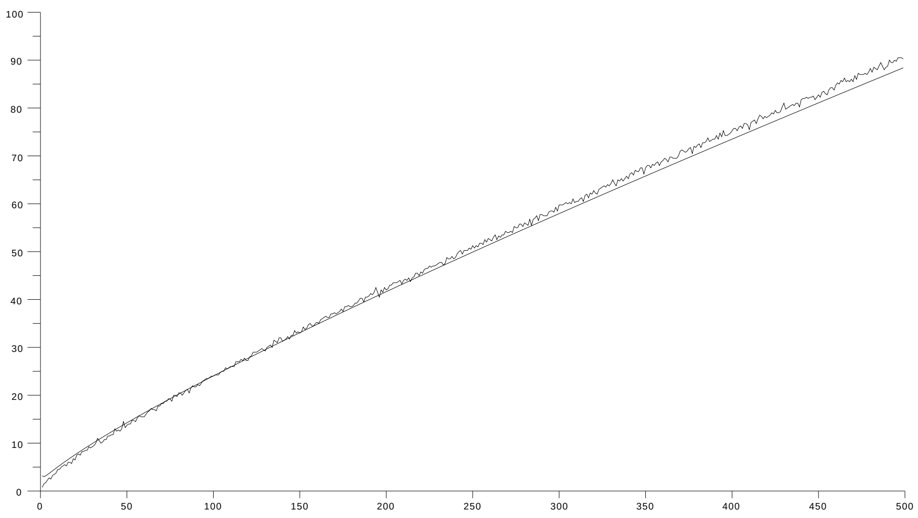

Figure 7.2: n vs $E[\chi(n, 0.67)]$

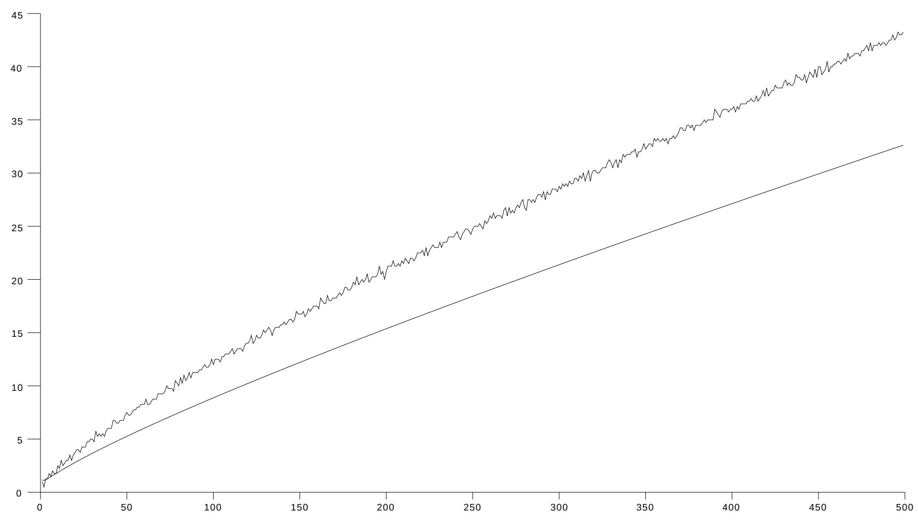

Figure 7.3: n vs $E[\chi(n, 0.33)]$

shown and in Table 7.5 mean results' mean is shown. We can see that both best and mean result is improving as the clique size is increasing. This denotes our greedy is a tunable greedy where to get better result we might increase the clique size. But it is important to note that this will take more time too.

Table 7.4: Best Result

$Vertex\ No.$	50	100	200
$3 - clique$	32.11	58.11	104.97
$4 - clique$	31.70	56.54	102.83
$5 - clique$	31.46	54.84	101.83

Table 7.5: Mean Result

$Vertex\ No$	50	100	200
$3 - clique$	34.48	60.83	108.32
$4 - clique$	33.70	59.41	105.96
$5 - clique$	33.61	59.11	105.12

We have also generated random weighted graphs with vertex number 50, 100, 200 and edge weight between $[0, 1]$, $[0, 2]$, $[0, 3]$, $[0, 4]$, $[0, 5]$. For each instances, we have generated 100 graph. That is 100 graphs are generated on 50 vertices where edge weights belong to the range $[0, 1]$ and so on. For each graph we run both our algorithm and the probabilistic greedy algorithm (PGA) [1] 10 times and measured the best span. We then take the mean over those 100 best results and shown in Table 7.6. It can be seen from the table that the span obtained by our algorithm is better than the PGA algorithm.

7.3 Results on Benchmark Instances

We have simulated our proposed algorithm over all the benchmark instances introduced in Section 2. The results are shown in Tables 7.7 and 7.8 for the problems 1 to 11. In Table 7.7 the computation time taken by our proposed algorithm is compared with other existing algorithms. Table 7.8 compares the span obtained by our algorithm with other existing algorithms. The row *Lower Bound* in Table 7.8 corresponds to the lower bound for each of the problems as reported in [18, 8, 19]. The rows *Best Results*, *Mean Results* and *Mean Gaps* represent the best result, average result of 100 runs and the deviation from optimality in percentage respectively obtained by our proposed algorithm. The results of problems 12 through 14 are not shown in the table. These problems are optimally solved by [1] and [3] in 0.1 and 0.23 seconds respectively. Our algorithm takes a fraction of milliseconds

Table 7.6: Comparison with PGA

$Vertex\ No$	$Edge\ weight$	$Mean\ span\ of\ our\ algorithm$	$Mean\ span\ of\ PGA$
50	$[0, 1]$	9.84	10.94
50	$[0, 2]$	20.74	23.36
50	$[0, 3]$	32.29	36.82
50	$[0, 4]$	43.69	49.86
50	$[0, 5]$	55.25	63.73
100	$[0, 1]$	17.08	19.00
100	$[0, 2]$	37.06	41.61
100	$[0, 3]$	58.02	65.21
100	$[0, 4]$	79.53	90.43
100	$[0, 5]$	101.64	115.76
200	$[0, 1]$	29.93	33.08
200	$[0, 2]$	65.57	75.52
200	$[0, 3]$	104.94	115.69
200	$[0, 4]$	144.34	160.65
200	$[0, 5]$	184.51	211.97

to produce the optimal result. For all problems we run 100 iterations of our algorithm and report the mean span. We get optimum result within 10 iterations for all problems except problems 2 and 6. For problems 2 and 6 our algorithm produces 428 and 260 channels but the computation times taken are 1.71 and 0.53 seconds only.

We also simulate our algorithm for 15, 30, 40 vertex benchmark instances as described by [1], where $s_1 = 3$ $s_2 = 2$ $s_3 = 1$. We run our algorithm for 3-clique 1000 times and report the best value. Both time and the obtained span are reported in Table 7.9. The time and span corresponding to the PGA algorithm are taken from [1]. Thus from this analysis we can conclude that our algorithm takes reasonably small computation time but produces span very close to the optimality and thus very much suitable for a real time application.

Table 7.7: Computation time comparisons between the existing cap algorithms and our approach

$Problems$	1	2	3	4	5	6	7	8	9	10	11
$Proposed\ Algotithm$	$0.95\ s$	$1.71\ s$	$1.28\ s$	$1.35\ s$	$0.19\ s$	$0.53\ s$	$0.27\ s$	$0.29\ s$	$0.02\ s$	$1.01\ s$	$1.77\ s$
(2013)[19]	$4\ ms$	$4\ ms$	$4\ ms$	$3\ ms$	$4\ ms$	$5\ ms$	$4\ ms$	$4\ ms$	$4\ ms$	$4\ ms$	$4\ ms$
(2006)[18]	$< 1\ s$	$10 - 20\ s$	$< 1\ s$	$< 1\ s$	$3 - 4\ s$	$10 - 20\ s$	$< 1\ s$	$3 - 4\ s$	$< 1\ s$	$< 1\ s$	$3 - 4\ s$
(2006)[6]	$few\ s$	$10\ h$	$few\ s$	$few\ s$	$-$	$10\ h$	$-$	$few\ s$	$-$	$-$	$-$
(2003)[17]	$2 - 5\ s$	$8 - 12\ h$	$0.5 - 1.0\ s$	$6 - 12\ s$	$2 - 7\ s$	$8 - 12\ h$	$0.5 - 1.0\ s$	$6 - 17\ s$	$-$	$-$	$-$
(2001)[16]	$7.5\ s$	$9.5\ s$	$8.2\ s$	$11.1\ s$	$6.9\ s$	$7.7\ s$	$6.0\ s$	$10.2\ s$	$1.9\ s$	$24.5\ s$	$16.7\ s$
(2001)[20]	$< 1\ s$	$110 - 170\ s$	$< 1\ s$	$< 1\ s$	$< 1\ s$	$110 - 170\ s$	$< 1\ s$	$< 1\ s$	$< 1\ s$	$-$	$-$
(1999)[8]	$few\ s$	$8\ m$	$few\ s$	$few\ s$	$few\ s$	$10\ m$	$few\ s$	$few\ s$	$-$	$-$	$-$
(1998)[9]	$-$	$-$	$-$	$-$	$-$	$596790\ s$	$19365\ s$	$89196\ s$	$-$	$-$	$-$
(1996)[23]	$< 1\ s$	$200 - 300\ s$	$< 1\ s$	$< 1\ s$	$< 1\ s$	$200 - 300\ s$	$< 1\ s$	$< 1\ s$	$< 1\ s$	$-$	$-$

Table 7.8: Comparisons of required bandwidth between the existing cap algorithms and our approach

Problems	1	2	3	4	5	6	7	8	9	10	11
Lower Bounds	380	426	532	532	220	252	308	308	72	308	70
Best Results	380	427	532	532	220	259	308	308	72	308	70
Mean Results	380.17	438.16	532.21	532.06	220.99	264.78	308.07	310.29	72	309.75	70
Mean Gaps(%)	0.05	2.85	0.04	0.19	0.45	5.05	0.02	0.74	0.00	0.24	0.00
(2013)[19]	381	448	533	534	221	267	308	311	72	309	73
(2006)[18]	380	426	532	532	220	252	308	308	72	308	70
(2006)[6]	380	431	532	532	–	252	–	308	–	–	–
(2003)[17]	380	426	532	532	220	252	308	308	–	–	–
(2001)[16]	380	462	532	532	220	272	308	308	72	308	78
(2001)[20]	380	426	532	532	220	253	308	308	72	–	–
(1998)[8]	380	426	532	532	220	252	308	308	–	–	–
(1998)[9]	–	–	–	–	220	267	–	308	–	–	–
(1996)[23]	380	432	532	532	220	262	308	308	72	–	–

Table 7.9: Benchmark problem with 15,30,45 vertex graph

Vertex No	15	30	40
Our Approach	16	24	18
PGA	16	28	20
Time			
Our Approach	$0.07s$	$0.32s$	$0.31s$
PGA	$1s$	$1s$	$1s$

Chapter 8

Conclusion

We have presented a novel probabilistic greedy algorithm to solve the CAP. The CAP is represented by means of a CAP graph. The proposed algorithm is based on first finding a probabilistic ordering and then assigning frequencies to them using frequency exhaustive strategy with forced assignment following that obtained ordering. Next a compression phase is iteratively applied to reduce the span as far as possible. Finally a smoothing phase is applied to increase the channel utilization. We have simulated our algorithm on well-known benchmark instances and notice that it gives close to optimum result in a very small execution time even for the most difficult benchmark instances. From the existing literature we saw that there are heuristics which take very high execution time to reach the optimum while some others take small execution time but results a span far from the optimality. Our algorithm takes reasonably small computation time but produces span very close to the optimality and thus very much suitable for a real time application.

Also we have generalized our approach. In the generalization we see that the span can be reduced at the cost of more execution time. Depending on the application, if bandwidth minimization is more important than the computation time we can go for higher order clique to get better result. We can use different clique size for the generalized forced assignment and generalized compression phase. So we give a tunable greedy which can be used according to the type of applications.

Bibliography

[1] A. Amiri, “A probabilistic greedy algorithm for channel assignment in cellular radio networks”, *IEEE Transactions on Communications*, Vol. 58, No. 11, pp. 3286-3295, 2010.

[2] G. Wang and N. Ansari, “Optimal broadcast scheduling in packet radio networks using mean field annealing,” *IEEE Journal on Selected Areas in Communications,* Vol. 15, No. 2, pp. 250-260, 1997.

[3] S. Menon, “A sequential approach for optimal broadcast scheduling in packet radio networks”, *IEEE Transactions on Communications*, Vol. 57, No. 3, pp. 764-770, 2009.

[4] D. Achlioptas and A. Naor, “The two possible values of the chromatic number of a random graph”, *Proc. of the 36th annual ACM symposium on Theory of computing*, STOC 2004.

[5] D. Gozupek, G. Gen, and C. Ersoy, “Channel assignment problem in cellular networks: A reactive tabu search approach,” *In 24th International Symposium on Computer and Information Sciences (ISCIS)*, pp. 298-303, 2009.

[6] X. Fu, A. G. Bourgeois, P. Fan, and Y. Pan, “Using a genetic algorithm approach to solve the dynamic channel-assignment problem,” *Int. J. Mobile Communications*, Vol. 4, No. 3, 2006.

[7] M. Duque-Anton, D. Kunz, and B. Ruber, “Channel assignment for cellular radio using simulated annealing,” *IEEE Transactions on Vehicular Technology*, Vol. 42, pp. 1421, Feb. 1993.

[8] D. Beckmann and U. Killat, “A new strategy for the application of genetic algorithms to the channel assignment problem,” *IEEE Transaction on Vehicular Technology*, Vol. 48, No. 4, pp. 1261-1269, July 1999.

[9] C. Y. Ngo and V. O. K. Li, “Fixed channel assignment in cellular radio networks using a modified genetic algorithm,” *IEEE Transaction on Vehicular Technology*, Vol. 47, No. 1, pp. 163-172, Feb. 1998.

[10] C. W. Sung, and W. S. Wong, “Sequential packing algorithm for channel assignment under co-channel and adjacent-channel interference constraint,” *IEEE Transaction on Vehicular Technology*, Vol. 46, pp. 676-686, Aug. 1997.

[11] E. Shamir and J. Spencer, “Sharp concentration of the chromatic number on random graphs Gn,p ,” Combinatorica 7 (1987), pp. 121?129.

[12] G. R. Grimmett and C. J. H. McDiarmid, “On coloring random graphs,” Math. Proc. Cam- bridge Philos. Soc. 77 (1975), pp. 313?324.

[13] Achlioptas, Dimitris, and Assaf Naor.“The two possible values of the chromatic number of a random graph.” Proceedings of the thirty-sixth annual ACM symposium on Theory of computing. ACM, 2004.

[14] C. E. C. Vieira, P. R. L. Gondim, C. A. Rodrigues, and J. L. Bordim. “A new technique to the channel assignment problem in mobile communication networks,” *The 19th IEEE International Symposium on In Personal, Indoor and Mobile Radio Communications (PIMRC)*, pp. 1-5, 2008.

[15] N. Funabiki and Y. Takefuji, “A neural network parallel algorithm for channel assignment in cellular radio network,” *IEEE Transaction on Vehicular Technology*, Vol. 41, pp. 430-437, Nov. 1992.

[16] G. Chakraborty, “An efficient heuristic algorithm for channel assignment problem in cellular radio networks,” *IEEE Transaction on Vehicular Technology*, Vol. 50, No. 6, pp. 1528-1539, Nov. 2001.

[17] S. C. Ghosh, B. P. Sinha and N. Das, “Channel assignment using genetic algorithm based on geometric symmetry,” *IEEE Transaction on Vehicular Technology*, Vol. 52, No. 4, pp. 860-875, July 2003.

[18] S. C. Ghosh, B. P. Sinha and N. Das, “Coalesced CAP : An improved technique for frequency assignment in cellular networks”, *IEEE Transaction on Vehicular Technology*, Vol. 55, pp. 640-653, March 2006.

[19] G. K. Audhya, K. Sinha, K. Mandal, R. Dattagupta, S. C. Ghosh, B. P. Sinha, “A New Approach to Fast Near-Optimal Channel Assignment in Cellular Mobile Networks”, *IEEE Transaction on Mobile Computing*, Vol. 12, No. 9, pp. 1814-1827, 2013.

[20] Roberto Battiti, Alan Bertossi, and Daniela Cavallaro, “A Randomized Saturation Degree Heuristic for Channel Assignment in Cellular Radio Networks,” *IEEE Transaction on Vehicular Technology,* Vol. 50, No. 2, pp. 364-374, Mar. 2001.

[21] Dong-Wan Tcha, June-Hyuk Kwon, Taek-Jin Choi, and Se-Hyun Oh, "Perturbation-Minimizing Frequency Assignment in a Changing TDMA/FDMA Cellular Environment," *IEEE Transaction on Vehicular Technology,* Vol. 49, No. 2, pp. 390-396, March 2000.

[22] J. S. Kim, S. H. Park, P. W. Dowd, and N. M. Nasrabadi, "Cellular radio channel assignment using a modified Hopfield network," *IEEE Transaction on Vehicular Technology,* Vol. 46, pp. 957-967, Nov. 1997.

[23] W. Wang and C. K. Rushforth, "An adaptive local-search algorithm for the channel-assignment problem (CAP)," in *IEEE Transaction on Vehicular Technology*, Vol. 45, No. 3, pp. 459-466, August 1996.