Solving Frequency Assignment Problems via Tree-Decomposition

Arie M.C.A. Koster[1,2] Stan P.M. van Hoesel[1] Antoon W.J. Kolen[1]

May 6, 1999

Abstract

In this paper we describe a computational study to solve hard frequency assignment problems (FAPs) to optimality using a tree decomposition of the graph that models interference constraints. We present a dynamic programming algorithm which solves FAPs based on this tree decomposition. We show that with the use of several dominance and bounding techniques it is possible to solve small and medium-size real-life instances of the frequency assignment problem to optimality. Moreover, with an iterative version of the algorithm we obtain good lower bounds for large-size instances within reasonable time and memory limits.

1 Introduction

The Frequency Assignment Problem (FAP) has two basic structural properties: the limited availability of frequencies to be assigned to wireless connections, and signal interference between connections for some combinations of frequencies. Practical applications range from military communication and television broadcasting to (the most popular example) mobile telephone communication. This diversity has not only resulted in many different models, but also in many different types of instances. The model we consider is fairly general in the sense that most variants of the FAP can be transformed to it. It consists of a set of antennas that are all to be assigned a frequency from an antenna-dependent set of available frequencies, the domain. In some applications certain frequencies are favored over others. We model this by introducing a penalty on each frequency from the domain of an antenna. For pairs of antennas specific combinations of frequencies may interfere, resulting in loss of quality of the reception of the signals. This loss of quality is measured and penalized with an amount related to the level of interference. For each pair of antennas, the penalties for all possible combinations of frequencies are stored in a penalty matrix. The penalty matrices have a structure that is often useful in solution methods, namely that frequencies within a given distance have a high penalty, and frequencies at larger distances have no penalty. The pair-wise relationship between the antennas allows for the following graph model: the vertices represent the antennas, each antenna pair

[1]Dept of Quantitative Economics, Maastricht University, P.O.Box 616, 6200 MD Maastricht, The Netherlands.
[2]e-mail: A.Koster@KE.UniMaas.NL; home page: http://www.unimaas.nl/~akoster/

with a nonzero penalty matrix is connected by an edge. This graph is called the constraint or interference graph. The standard objectives are to find a frequency plan that minimizes the sum of the vertex and edge-penalties, or that minimizes the maximum penalty.

The determination of the penalties slightly depends on the application. In mobile phone networks the area where signals interfere may also vary. In that case, the penalty is not only related with the level of interference, but also with the size of the area in which the interference is measured. Minimization of the changes in an existing frequency plan can be taken into account via a vertex penalty on the other frequencies for an antenna. Also the number of frequencies that has to be assigned to an antenna may vary per application. In mobile networks antennas are part of larger units, such as sites. These units are sometimes treated like antennas so that multiple frequencies are to be assigned to each site. In some applications the objective is not to minimize the sum of the penalties, but to answer the question whether there exists an assignment without penalty. We refer to Hale [10] for an overview of models for the FAP with other objectives. In this paper we concentrate on the objective to minimize the total penalty inflicted by a plan.

The FAP is, in general, hard to solve, due to its close relation to the vertex coloring problem. Namely, a special case of the FAP is the one, in which equal frequencies for vertices adjacent in the constraint graph are penalized solely. Therefore, many heuristic approaches have been suggested using the known methods in operations research and artificial intelligence, like simulated annealing (cf. Hurkens and Tiourine [11]), tabu search (cf. Castelino, Hurley and Stephens [7]) and genetic algorithms (Kolen [14]). A comparison of these techniques on a specific set of data can be found in [19]. In Borndörfer et al. [6] both heuristics based on graph coloring, and local search techniques are described. In this paper we concentrate on finding exact solutions, or (second best) on finding good lower bounds for the FAP. To obtain lower bounds for this problem, Hurkens and Tiourine [11] use nonlinear programming techniques. Good lower bounds are only obtained for very special cost structures and fairly simple constraint graphs. An exact solution technique has been studied by Koster, van Hoesel and Kolen [15]. They investigate the polyhedral approach which can also be used to obtain lower bounds. This approach only works within reasonable time for problems with a limited number of frequencies available for every antenna. Recently, Jaumard et al. [13] have tried column generation for solving the FAP. Column generation seems to be able to solve to optimality only small instances, but generates very good solutions during the process. Aardal et al. [1] use a branch and cut algorithm to solve the FAP in which a solution without penalty is to be find, whereas in Janssen and Kilakos [12], lower bounds for the minimum span frequency assignment problem are obtained via polyhedral analysis of the problem. For an overview about exact approaches for the frequency assignment problem we refer to Aardal et al. [2].

Forced by the limited success of the exact solution methods so far, we tried to exploit the structure of the constraint graph more directly in our approach. Instances of the FAP have a geographical nature, since each antenna is placed in a two-dimensional map. Moreover, this geography influences interference, since pairs of antennas have no interference if their distance is far enough. Finally, concentrations of antennas are found in densely populated areas. These areas are connected with one another with a limited number of edges. This led us to believe that many instances have a constraint graph with a *tree-like structure*, and thus may be solved using a *tree decomposition* of the constraint graph with small *treewidth*. The notions treewidth and

tree decomposition are introduced by Robertson and Seymour [18] in their fundamental work on graph minors. Besides the major role they play in graph theory, many NP-hard problems on graphs have been shown to be solvable in polynomial (linear) time on graphs with bounded treewidth (see Bodlaender [4] for an overview). We used this idea, together with sophisticated processing techniques, on a set of instances for which the previous techniques generated only few significant results, i.e., only for a small set of instances non-trivial lower bounds were computed. We are now able to solve many of these instances to optimality. Moreover, in an iterative version of our algorithm we are able to generate good lower bounds on the very difficult instances. The algorithm is applicable on many instances. The only serious limitation is the treewidth of the constraint graph. Finally, we mention that the FAP is a partial constraint satisfaction problem with binary relations (PCSP). It seems likely due to the generic nature of the FAP, that our techniques are also applicable to other PCSPs.

The main purpose of this paper is twofold. In the first place, our goal is to find benchmarks for a set of publicly available FAPs. Secondly, our purpose is to show that the concept of tree decomposition is not only of theoretical value, but can really used to solve combinatorial optimization problems to optimality. We do not have the intention to demonstrate this method as *the* method to solve FAPs. For that purpose the method is not competitive compared with available heuristics.

The remainder of this paper is organized as follows. In Sections 2 and 3 we respectively model the FAP in detail, and we introduce the graph theoretic concepts we use in the paper, such as treewidth. In Section 4 we describe the heuristic method we use to obtain a tree decomposition of the constraint graph, and in Section 5 we propose the dynamic programming algorithm based on the tree decomposition of the constraint graph. The practical utility of the algorithm can be improved via the use of (pre)processing techniques, which are described in Section 6. We present an iterative extension of the algorithm that provides lower bounds for the original problem in Section 7. The computational results obtained with these methods are the topic of Section 8.

2 Problem Description

A FAP is defined by the quadruple (G, D, p, q), where $G = (V, E)$ is the constraint graph. The set $D = \{D_v : v \in V\}$ is the collection of all domains. For each of the n vertices $v \in V$ in the graph the domain D_v contains the frequencies available for that vertex. The last two components of the FAP are two penalty functions p and q. For each pair of adjacent vertices and corresponding choice of frequencies, the function p determines the interference penalty. The function q denotes the level of preference for all domain elements. The functions p and q are called the edge-penalty function, and the vertex-penalty function, respectively. The objective of the problem is to select from every domain D_v exactly one element in such a way that the total sum of the edge- and vertex-penalties is minimized.

The FAP is formulated as a binary linear programming problem using the following binary

variables for all $v \in V$, $d_v \in D_v$

$$y(v, d_v) = \begin{cases} 1 & \text{if } d_v \in D_v \text{ is selected} \\ 0 & \text{otherwise} \end{cases}$$

and for all $\{v, w\} \in E$, $d_v \in D_v$, $d_w \in D_w$

$$z(v, d_v, w, d_w) = \begin{cases} 1 & \text{if } (d_v, d_w) \in D_v \times D_w \text{ is selected} \\ 0 & \text{otherwise} \end{cases}$$

The binary linear programming formulation then reads

$$\min \quad \sum_{\{v,w\} \in E} \sum_{d_v \in D_v} \sum_{d_w \in D_w} p(v, d_v, w, d_w) z(v, d_v, w, d_w) + \sum_{v \in V} \sum_{d_v \in D_v} q(v, d_v) y(v, d_v) \tag{1}$$

$$\text{s.t.} \quad \sum_{d_v \in D_v} y(v, d_v) = 1 \qquad \forall v \in V \tag{2}$$

$$\sum_{d_w \in D_w} z(v, d_v, w, d_w) = y(v, d_v) \qquad \forall \{v, w\} \in E, d_v \in D_v \tag{3}$$

$$z(v, d_v, w, d_w) \in \{0, 1\} \qquad \forall \{v, w\} \in E, d_v \in D_v, d_w \in D_w \tag{4}$$

$$y(v, d_v) \in \{0, 1\} \qquad \forall v \in V, d_v \in D_v \tag{5}$$

Constraints (2) restrict the selection of frequencies from each domain to one. Constraints (3) enforce that the combination of values selected for an edge should be consistent with the values selected for the vertices of that edge.

The NP-hardness of the FAP with domain sizes at least 3 follows from a reduction of the graph 3-colorability problem (cf. [8]). In Koster, van Hoesel and Kolen [15] it is proved with a reduction from Maximum Satisfiability that the FAP is NP-hard, even if all domains have size 2.

In the sequel of this paper we use the following notation. Let $N(v) = \{w \in V : \{v, w\} \in E\}$ denote the set of vertices adjacent to $v \in V$, whereas $N(S) = \{w \in V \setminus S : \exists_{v \in S} \{v, w\} \in E\}$ denotes the neighbors of the vertices in the subset $S \subseteq V$. Moreover, let $\delta(S, T)$ denote the set of all edges between the vertices in $S \subseteq V$ and $T \subseteq V$, i.e., $\delta(S, T) = \{\{v, w\} \in E : v \in S, w \in T\}$. We use $\delta(S)$ as short version of $\delta(S, V \setminus S)$. With $E[S]$ we denote all edges with both vertices in S, i.e., $E[S] = \delta(S, S)$. By $G[S] = (S, E[S])$ we denote the subgraph of $G = (V, E)$ induced by S.

3 Graph Theoretic Concepts

In this section we introduce the graph theoretic concepts used in our solution method. We define the notions tree decomposition and treewidth, together with some (well-known) properties of

these notions. We also define the concept separating vertex set, which will be used in the heuristic to construct a tree decomposition.

Before we introduce the notion of tree decomposition of a graph we start with the simpler notion of path decomposition (Robertson and Seymour [17]). A path decomposition decomposes the graph in a sequence $i = 1, \ldots, r$ of subgraphs induced by subsets $X_i \subseteq V$. All vertices and edges have to be in at least one subgraph. Moreover, if a vertex is part of two induced subgraphs, then all the subgraphs in between these two in the sequence should also contain this vertex. Or equivalently, the subgraphs for which the vertex sets contain a certain vertex should be a subsequence of the total sequence. The width of a path decomposition is given by the maximum size of the vertex sets of the subgraphs minus one. The pathwidth of a graph G is the minimum width over all path decompositions of G. Formally,

Definition 3.1 (Robertson and Seymour [17]) *Let $G = (V, E)$ be a graph. A path-decomposition is a sequence $X_1, \ldots, X_r$ of subsets of V, such that*

(i). $\bigcup_{i=1,\ldots,r} X_i = V$,

(ii). for every edge $\{v, w\} \in E$, there is an $i \in I$ with $v \in X_i$ and $w \in X_i$, and

(iii). for all $i, j, k \in \{1, \ldots, r\}$, if $i < j < k$, then $X_i \cap X_k \subseteq X_j$.

The width of a path decomposition is $max_{i=1,\ldots,r}|X_i| - 1$. The pathwidth of a graph G, denoted by $pw(G)$, is the minimum width over all possible path decompositions of G.

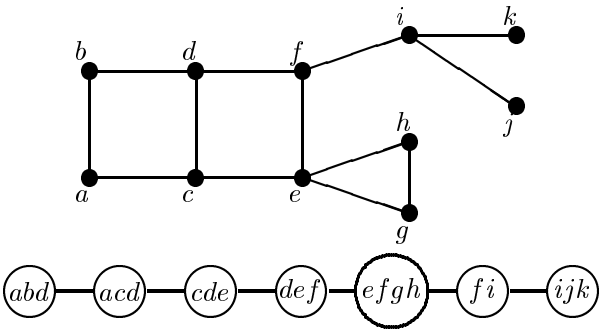


Figure 1: Example of a graph and path decomposition with width 3

In Figure 1 an example of a graph and an optimal path decomposition with width 3 is given. For special classes of graphs the pathwidth is known in advance (cf. [4]). For example, if the graph consists of a single path, the pathwidth is equal to one. For trees the pathwidth is $\mathcal{O}(d)$, where d is the length of the longest path in the tree. Robertson and Seymour developed a variant of the path decomposition concept called tree decomposition in [18]. Instead of a decomposition of the graph into a path, the graph is decomposed into a tree of induced subgraphs. The width of a tree decomposition is the maximum cardinality of the subgraphs minus one. Formally,

Definition 3.2 (Robertson and Seymour [18]) *Let $G = (V, E)$ be a graph. A tree-decomposition is a pair $(T, \mathcal{X})$, where $T = (I, F)$ is a tree with nodes I and edges F, and $\mathcal{X} = \{X_i : i \in I\}$ is a family of subsets of V, one for each node of T, such that*

(i). $\bigcup_{i \in I} X_i = V$,

(ii). for every edge $\{v, w\} \in E$, there is an $i \in I$ with $v \in X_i$ and $w \in X_i$, and

(iii). for all $i, j, k \in I$, if j is on the path from i to k in T, then $X_i \cap X_k \subseteq X_j$.

The width of a tree decomposition is $max_{i \in I}|X_i| - 1$. The treewidth of a graph G, denoted by $tw(G)$, is the minimum width over all possible tree decompositions of G.

The third condition of the tree decomposition is equivalent to the condition that for all $v \in V$, the set of nodes $\{i \in I : v \in X_i\}$ is connected in T. Note that, since each path decomposition is also a tree decomposition, $tw(G) \leq pw(G)$.

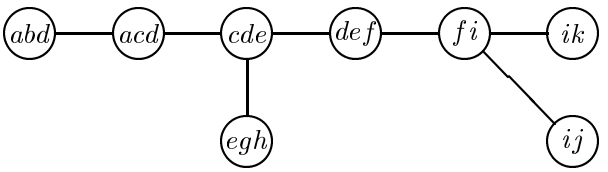


Figure 2: Example of a tree decomposition with width 2

In Figure 2 an optimal tree decomposition of the graph of Figure 1 is given. The width of this decomposition is 2. A connected graph has treewidth 1 if and only if the graph is a tree. The complexity of the construction of a tree decomposition (path decomposition) of minimal treewidth (pathwidth) is discussed in the next proposition.

Proposition 3.1

(i). The problem 'Given a graph $G = (V, E)$ and an integer k, is the treewidth (pathwidth) of G at most k' is NP-complete.

(ii). Given a constant integer k, the problem 'Given a graph $G = (V, E)$, is the treewidth (pathwidth) of G at most k' can be solved in polynomial time.

So, if the integer k is part of the input of the problem, the problem is NP-complete whereas it can be solved in polynomial time in case k is fixed. The NP-completeness results for treewidth and pathwidth are due to Arnborg, Corneil and Proskurowski [3]. An algorithm that solves the problem in polynomial time for constant k is given by Bodlaender [5]. However, this algorithm is exponential in k, and is therefore impractical for graphs with larger treewidth. Therefore, we use a heuristic to construct tree decompositions.

In a tree decomposition we can remove nodes for which the corresponding vertices form a subset of the vertices of another node. As a consequence, every tree decomposition can be transformed to a tree decomposition in which the vertex-sets of all internal nodes separate the constraint graph in at least two components, i.e., the vertices form a separating vertex set.

Definition 3.3 *An st-separating set of $G = (V, E)$ is a set $S \subseteq V \setminus \{s, t\}$ with the property that any path from s to t passes through a vertex of S. The minimal separating vertex set of G is given by the st-separating set with minimum cardinality over all combinations $\{s, t\} \notin E$.*

Note that the separating vertex sets in a tree decomposition are not necessarily minimal. The property that every internal node correspond with a separating vertex set forms the basis of our heuristic, which is the topic of the next section.

4 Construction of a Tree-Decomposition

Since the algorithm we want to use for solving FAPs heavily depends on the width of the tree decomposition of the constraint graph, we need a tree decomposition with small width. Finding a tree decomposition with optimal width is NP-hard. Therefore, we implemented a sequential improvement heuristic. The algorithm aims at decreasing the cardinality of the nodes in a given tree decomposition based on the property that the vertices that correspond to internal nodes of the tree are separating vertex sets in the graph. We try to replace a node in an existing tree decomposition by a number of new nodes for which the maximum cardinality is smaller than the cardinality of the original node. To achieve this goal, we search for small separating vertex sets. In Section 4.1 we describe the algorithm to find a minimum separating vertex set in a graph, whereas the heuristic itself is the topic of Section 4.2.

4.1 Minimum separating vertex set in a graph

For any combination of 2 non-adjacent vertices, the st-separating set with minimal cardinality can be found efficiently using Menger's theorem.

Theorem 4.1 (Menger [16]) *Given a graph $G = (V, E)$ and two distinct non-adjacent vertices $s, t \in V$, the minimum number of vertices in an st-separating set is equal to the maximum number of vertex-disjoint paths connecting s and t.*

So, we have to calculate the maximum number of vertex-disjoint paths. This problem is solvable in polynomial time by standard network flow techniques. First, we construct a directed graph $D = (V, A)$, in which each edge {v,w} is replaced by two arcs (v, w) and (w, v) both with weight ∞. Next, we construct an auxiliary directed graph D' by

- replacing each vertex v by two vertices v' and v'',

- redirecting each arc with head v to v',
- redirecting each arc with tail v to v'', and
- adding an arc from v' to v'' with weight 1.

Then, the minimum number of vertices in an st-separating set in G is equal to the minimum weight of an $s'' - t'$ cut in D'. So, if we calculate the minimum $s'' - t'$ cut for every combination $s, t \in V$, $\{s, t\} \notin E$, we obtain the minimum separating vertex set. Note that since the graph D' is a directed graph, we have to solve $\mathcal{O}(n^2)$ minimum cut problems. In other words, we cannot use the algorithm of Gomory and Hu [9], which solves the all pairs minimum cut problem for undirected graphs by solving only $\mathcal{O}(n)$ minimum cut problems.

4.2 Heuristic

The heuristic we use to obtain a tree decomposition can be described as follows. We start with the trivial tree decomposition in which we have one node corresponding to the complete graph. During the process we have a tree decomposition $(T, \mathcal{X})$. We select the node $i \in I$ with $|X_i|$ maximum. This node is replaced by $m + 1$ nodes $i_0, \ldots, i_m$ with vertex sets $X_{i_0}, \ldots, X_{i_m}$. The nodes $i_1, \ldots, i_m$ all are connected with i_0. Each node $k \in N(i)$ is connected to exactly one node $j \in \{i_0, \ldots, i_m\}$, such that all conditions of a tree decomposition are satisfied again.

The sets $X_{i_0}, \ldots, X_{i_m}$ are defined as follows. We construct a graph $G_i = (V_i, E_i)$ that consist of the induced subgraph $G[X_i]$ and the additional edges $\cup_{k \in N(i)} C(X_i \cap X_k)$, where $C(X)$ denotes a complete graph on the vertices X (i.e., $C(X)$ is a clique). If G_i is a complete graph, then $X_{i_0} := X_i$ and $m = 0$, i.e., we do not change the tree decomposition. If G_i is not a clique, then we calculate a minimum separating vertex set $S \subset V_i$. Let $Y_{i_1}, \ldots, Y_{i_m}$ be the vertex sets of the $m \geq 2$ components of $G_i[V_i \setminus S]$. We define $X_{i_0} := S$, and $X_{i_j} := Y_{i_j} \cup S$ for all $j \in 1, \ldots, m$. The set X_k has a non-empty intersection with at most one set Y_{i_j}, $j = 1, \ldots, m$: Let $v, w \in X_i \cap X_k$, then $\{v, w\} \in C(X_i \cap X_k) \subset E_i$, which implies that v and w cannot be separated by S. So, either $v, w \in S$ or $v, w \in Y_{i_j} \cup S$ for only one $j \in \{1, \ldots, m\}$. Therefore, we connect each neighbor $k \in N(i)$ with the node i_j, $j \in \{1, \ldots, m\}$ for which the intersection of X_k and Y_{i_j} is non-empty, or in case there is none with i_0. As a consequence, the new construction is a tree again (see Figure 3). In the new tree the conditions for a valid tree decomposition again

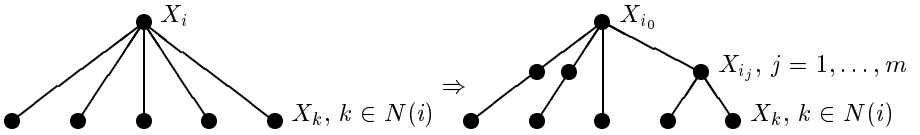


Figure 3: Improvement step of a tree decomposition

hold. Since $\cup_{j=0}^{m} X_{i_j} = (\cup_{j=0}^{m} Y_{i_j}) \cup S = X_i$ condition (i) is satisfied. To satisfy condition (ii) we have to prove that for each edge $\{v, w\} \in E[X_i]$ one of the new vertex sets $X_{i_0}, \ldots, X_{i_m}$ contains both vertices. If $v, w \in S$, then this is trivially true. Otherwise, suppose $v \in Y_{i_j}$ for

some $j \in \{1, \ldots, m\}$. If $w \in Y_{i_k}$, $k \neq j$, then S does not separate Y_{i_j} and Y_{i_k}; a contradiction. And thus, $w \in Y_{i_j} \cup S = X_{i_j}$. Condition (iii) states that all nodes in the tree that contain the same vertex v must form a subtree. We only need to check this for $v \in X_i$. If $v \in S$ then v is contained in all new nodes and the condition is trivially satisfied. Otherwise, let $v \in Y_{i_j}$ for some $j \in \{1, \ldots, m\}$. By construction, nodes $k \in N(i)$ and i_j are connected if X_k and Y_{i_j} intersect. Hence, all nodes that contain v form a subtree again.

Note that, if G_i is not a clique, then there exist vertices $v, w \in X_i$ with $\{v, w\} \notin E_i$. Thus $S = X_i \setminus \{v, w\}$ separates G_i in two components; $Y_{i_1} = \{v\}$ and $Y_{i_2} = \{w\}$. So, $\max\{|Y_{i_1} \cup S|, |Y_{i_2} \cup S|\} = |X_i| - 1 < |X_i|$. As a consequence, the width of the tree decomposition may decrease. Figure 4 shows the heuristic in a flowchart.

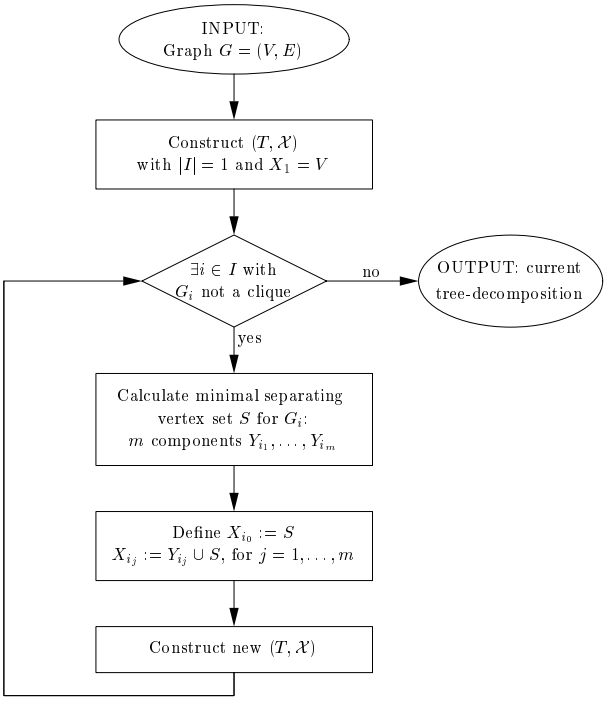


Figure 4: Heuristic for construction of a tree decomposition

5 Dynamic Programming Algorithm

The algorithm that solves the FAP in polynomial time (given that the treewidth is at most a constant k) is based on the following idea. Let $S \subset V$ be a separating vertex set of G with $G[V \setminus S] = G[V_1] \cup G[V_2]$. Then the optimal assignment in V_1 (or V_2) only depends on the assignment in S. So, given an assignment of S the problem decomposes in two FAPs on $G[V_1]$ and $G[V_2]$. Thus, the FAP can be solved by solving the two FAPs on $G[V_1]$ and $G[V_2]$ for all possible assignments in S. This idea can be formulated as a dynamic programming algorithm

using a tree decomposition of the graph. For every internal node $i \in I$, X_i is a separating vertex set, which implies that given an assignment for X_i, the FAP decomposes in smaller FAPs for every branch in the tree.

Before we describe the algorithm in more detail, we first introduce some additional notation. In the sequel of the paper we assume that the tree is rooted and binary. Let $Y_i = \{v \in V : \exists j \in I, j \text{ descendant of } i \text{ and } v \in X_j\}$ denote the set of vertices that is represented by the subtree rooted at node i. Given a subset $S \subseteq V$, we denote with $d_S = (d_v)_{v \in S}$ an assignment of domain elements $d_v \in D_v$ for every vertex $v \in S$. Similar, D_S denote the complete set of all assignments for a set S.

Now, we can describe the dynamic programming algorithm as follows. In a bottom-to-top way we compute for every node $i \in I$ all assignments for the subset Y_i, D_{Y_i}. Starting with a leaf $i \in I$ of the tree, the algorithm stores all assignments for the vertices in X_i. The computation of all assignments takes $\mathcal{O}(\Pi_{v \in X_i} |D_v|) = \mathcal{O}(d^{|X_i|})$ time, where $d = \max_{v \in V} |D_v|$. Next, given all assignments for two nodes $j, k \in I$ with common predecessor $i \in I$, we can compute all assignments Y_i by combining every assignment of Y_j, every assignment of Y_k that has the same assignment for the vertices in $X_j \cap X_k$, and every assignment of domain elements to the vertices in $X_i \setminus (X_j \cup X_k)$. However, since X_i is a separating vertex set in the graph, we do not have to store all assignments for the vertices in Y_i, but only the assignments that differ for the vertices in X_i. For an assignment of the vertices in X_i, we only have to store the best assignment for the vertices in $Y_i \setminus X_i$. In other words, we have to store at most $\Pi_{v \in X_i} |D_v|$ assignments for node $i \in I$ instead of $\Pi_{v \in Y_i} |D_v|$ assignments to obtain the overall optimal solution. The computation of these assignments can be done in $\mathcal{O}(\Pi_{v \in X_i \cup X_j \cup X_k} |D_v|) = \mathcal{O}(d^{|X_i|+|X_j|+|X_k|})$. Finally, for the root node $r \in I$ of the tree T, $Y_r = V$, and so we only have to store one solution which gives the desired optimal solution for the problem. The overall computation time of this algorithm is given by $\mathcal{O}(nd^{3k})$, where k is the width of the tree decomposition $(T, \mathcal{X})$ of G that is used. So, for graphs with treewidth bounded by a constant k, this algorithm solves the FAP in time polynomial in n and d, but exponential in k. In Figure 5 the algorithm is represented in a flowchart, where we assume that the nodes are numbered $1, \ldots, |I|$ in a topological order from top to bottom.

The performance of the algorithm highly relies on additional techniques to reduce the size of the sets of assignments D_{Y_i}. These techniques are described in the next section.

6 Reduction Techniques

Quick ways to remove vertices and edges from the constraint graph or to remove frequencies from the domains of the vertices may speed up any solution technique for the FAP applied afterwards. Our technique for solving the FAP, a dynamic programming algorithm based on the tree decomposition of G, computes all non-redundant assignments for subsets of vertices. The number of different assignments grows exponentially with the cardinality of the subset, which makes the need for good reduction techniques evident. In this section we present several such techniques. All are based on the following paradigm for extending partial feasible solutions:

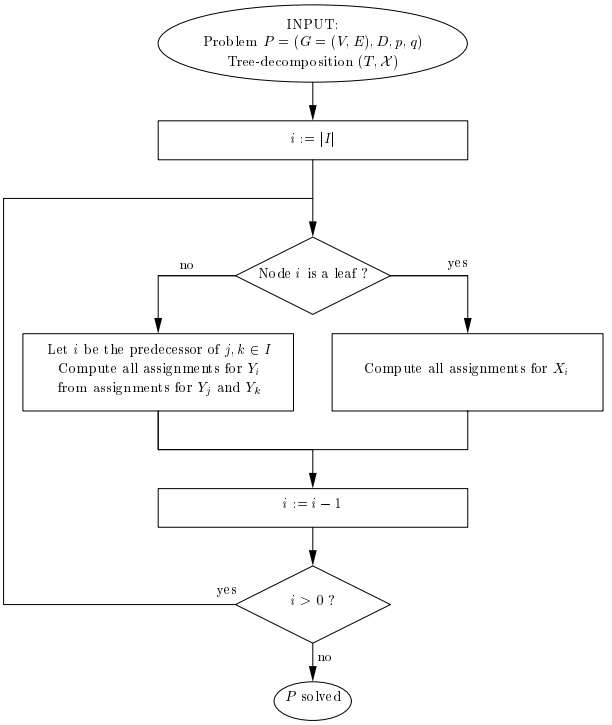


Figure 5: Dynamic programming algorithm

A partial feasible solution can be extended to an optimal solution *only if* the extension itself is optimal with respect to the partial feasible solution. In other words, if a partial feasible solution is not extended optimally, the resulting feasible solution is certainly not optimal.

In the next subsection we use this paradigm directly to remove vertices, or replace them by edges. In Subsection 6.2 we present a penalty shifting procedure, which is mainly used to obtain lower bounds on the value of the instances, but can sometimes remove edges from the constraint graph as well. In Subsection 6.3, we present techniques to remove frequencies from the domains of vertices, and to remove non-optimal partial feasible solutions. This is done in two ways: by using upper bounding techniques, and by using dominance criteria.

6.1 Constraint graph reduction

In this subsection we describe how we can remove vertices $v \in V$ with $|D_v| = 1$ or $|N(v)| \leq 2$ from G. First of all, for vertices v with $D_v = \{d_v^*\}$ we do not have a choice for the frequency. Therefore v can be removed from the constraint graph, provided that $q(v, d_v^*)$ is added to the solution value, and that for every $w \in N(v)$, $d_w \in D_w$ the penalty $p(v, d_v^*, w, d_w)$ is added to the vertex penalty $q(w, d_w)$. Vertices with degree zero can also be removed from the constraint graph. For a vertex $v \in V$ with $|N(v)| = 0$, the optimal choice of a frequency is $\arg\min_{d_v \in D_v} q(v, d_v)$. So, the vertex can be removed from the graph, provided that the value of the optimal solution

in the remaining problem will be increased with $\min_{d_v \in D_v} q(v, d_v)$.

Next, let $v \in V$ be a vertex with $|N(v)| = 1$, and let $N(v) = \{w\}$. Consider a partial solution in which v is the only vertex without a frequency assigned to it. We should assign a frequency to v, that has minimal penalty with respect to the partial solution. To do so, we only need to consider the frequency assigned to w, say d_w^*, since the other vertices are not connected to v in G, and, thus, do not influence the penalty incurred by any choice of frequency for v. Therefore, it suffices to compute the smallest penalty incurred by the frequencies of v, i.e., $\min_{d_v \in D_v}\{q(v, d_v) + p(v, d_v, w, d_w^*)\}$, and extend the partial feasible solution with a frequency d_v^* that attains this minimum. Although d_w^* may differ among all partial solutions, we can determine the best extension of any partial feasible solution beforehand by, for all $d_w \in D_w$, computing the value

$$q'(w, d_w) = \min_{d_v \in D_v}\{q(v, d_v) + p(v, d_v, w, d_w)\}$$

and subsequently adding $q'(w, d_w)$ to $q(w, d_w)$. This, in effect, adds to each d_w the optimal choice in D_v at the beginning of the algorithm, allowing us to remove the vertex v and the edge $\{v, w\}$ from the instance. At the end of the algorithm an optimal solution found for the problem instance restricted to $G[V \setminus \{v\}]$, can then be extended by selecting the optimal choice $d_v^* \in D_v$ given the chosen frequency d_w^* of w.

We can generalize this idea to vertices with degree two as follows. Let v be such a vertex, and let $N(v) = \{u, w\}$. To extend a partial solution in which v is the only node without a frequency, we should assign a frequency to v, that is optimal with respect to the frequencies of u and w. Let d_u^* and d_w^* be the selected frequencies. We then select $d_v^* = \arg\min_{d_v \in D_v}\{p(u, d_u^*, v, d_v) + q(v, d_v) + p(v, d_v, w, d_w^*)\}$. Again, we can do this beforehand by, for all $d_u \in D_u, d_w \in D_w$, computing the value

$$p'(u, d_u, w, d_w) = \min_{d_v \in D_v}\{p(u, d_u, v, d_v) + q(v, d_v) + p(v, d_v, w, d_w)\}$$

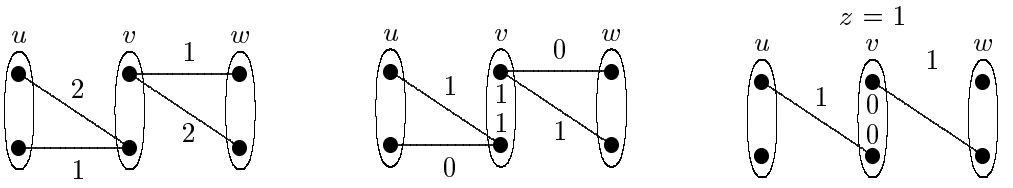
and subsequently adding $p'(u, d_u, w, d_w)$ to $p(u, d_u, w, d_w)$. This, in effect, adds to each combination $\{d_u, d_w\}$ the optimal choice in D_v, allowing us to remove the vertex v and its two incident edges from the instance. Note that possibly the edge $\{u, w\}$ may have to be inserted in the constraint graph.

We can repeat the reduction process until all vertices with degree at most two are removed.

6.2 Penalty shifting - Lower bounding

In this subsection we present a technique to obtain a lower bound on the optimal value of the instances by shifting penalties from edges to vertices and back, and from vertices to the objective and back. We first illustrate the technique by the example in Figure 6(a). We have three vertices, each with 2 domain elements. The non-zero edge-penalties are given by edges. We can

transform this part of the instance by moving penalty from the penalty matrix to the penalties on frequencies (Figure 6(b)), and even from the frequencies to the objective (Figure 6(c)).



(a) original (b) shift to vertex v (c) shift to objective

Figure 6: Example shifting penalties

If for an edge $\{v, w\} \in E$ we have a penalty matrix such that given $d_v^* \in D_v$ for all $d_w \in D_w$, $p(v, d_v^*, w, d_w) > 0$ then by model equality (3) we can decrease these penalties, and simultaneously increase $q(v, d_v^*)$ with the same amount. The same procedure works on vertices. Suppose that we have a positive penalty $q(v, d_v)$ for all $d_v \in D_v$. Then by (2) we can decrease the penalty $q(v, d_v)$ with the minimum vertex penalty and add the same value to the objective. The condition that all penalties should be nonnegative is not really crucial, but allows us to maintain a lower bound on the objective value.

A special case are penalty matrices with the property that $p(v, d_v, w, d_w) = \bar{q}(v, d_v) + \bar{q}(w, d_w)$, i.e., the elements are the sum of values corresponding to the rows and columns. Then we can reduce all edge-penalties to zero, and thus remove the edge from the constraint graph by shifting all edge-penalties to the frequencies of the two corresponding vertices.

6.3 Domain reduction

In this section we devise methods to reduce the number of partial feasible solutions to the ones that are candidates to be used in optimal solutions. We describe two ways of doing so, namely upper bounding (in Section 6.3.1), and dominance (in Section 6.3.2).

6.3.1 Upper bounding

Upper bounding in its simplest form is performed on vertices as follows. Consider a vertex v and its neighbors $N(v)$. We want to derive an upper bound $u(v, \delta(v))$ on the total penalty incurred by node v in the optimal solution of the FAP, i.e., an upper bound on the vertex-penalty of v and the edge-penalties of the edges incident with v.

Consider an arbitrary partial solution $d_{N(v)}^* \in D_{N(v)}$. Then we compute the frequency for v

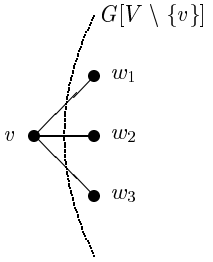
with the lowest penalty:

$$P(d^*_{N(v)}) = \min_{d_v \in D_v} \left\{ q(v, d_v) + \sum_{w \in N(v)} p(v, d_v, w, d^*_w) \right\}$$

Among all possible choices for $d^*_{N(v)} \in D_{N(v)}$ we take the one with highest penalty, i.e.

$$u(v, \delta(v)) = \max_{d^*_{N(v)} \in D_{N(v)}} P(d^*_{N(v)})$$

Then the value $u(v, \delta(v))$ is certainly an upper bound on the penalty incurred by an optimal choice of frequency for v.



D_v	$q(v, d_v)$
1	0
2	0
3	0

$p(v, d_v, w, d_{w_1})$

	D_{w_1}	
D_v	1	2
1	2	0
2	0	2
3	1	1

$p(v, d_v, w, d_{w_2})$

	D_{w_2}	
D_v	1	2
1	0	2
2	2	0
3	1	1

$p(v, d_v, w, d_{w_3})$

	D_{w_3}	
D_v	1	2
1	1	0
2	0	1
3	0	0

Figure 7: Example upper bounding and dominance

We illustrate this upper bounding technique with the following example, see Figure 7. Let v be a vertex in the constraint graph G. Its domain contains three frequencies: 1, 2, and 3. It is connected to three vertices w_1, w_2, and w_3 each of which has two frequencies: 1 and 2. For all $d_v \in D_v$, the total penalty is $q(v, d_v) + \sum_{w \in \delta(v)} p(v, d_v, w, d^*_w)$ where d^*_w is the frequency chosen for w. In Table 1 we have computed for any combination of $(d_{w_1}, d_{w_2}, d_{w_3})$ the best frequency d^*_v for v, i.e., the one such that the total penalty is minimal among all possible frequencies for v. Table 1 shows that for this example the upper bound is 2, the maximum of the last column. In general, though not in this example, any frequency $d_v \in D_v$ with $q(v, d_v) > u(v, \delta(v))$ can be removed from D_v.

For arbitrary $v \in V$, we can compute this upper bound by solving an integer linear program. For all $w \in N(v)$, $d_w \in D_w$ we introduce a binary variable

$$y(w, d_w) = \begin{cases} 1 & \text{if } d_w \in D_w \text{ is assigned to } w \in N(v) \\ 0 & \text{otherwise} \end{cases}$$

If the variable z denotes the actual upper bound, the integer linear program reads as

$$u(v, \delta(v)) = \max \quad z \tag{6}$$

d_{w_1}	d_{w_2}	d_{w_3}	$q(v,d_v)+\sum_{w\in N(v)} p(v,d_v,w,d_w)$			
			$d_v=1$	$d_v=2$	$d_v=3$	best
1	1	1	5	0	2	0
1	1	2	4	1	2	1
1	2	1	3	2	2	2
1	2	2	2	3	2	2
2	1	1	3	2	2	2
2	1	2	2	3	2	2
2	2	1	1	4	2	1
2	2	2	0	5	2	0

Table 1: Penalties example upper bounding and dominance

$$\text{s.t.} \quad z \le q(v,d_v) + \sum_{w\in N(v)} \sum_{d_w\in D_w} p(v,d_v,w,d_w) y(w,d_w) \forall d_v \in D_v \tag{7}$$

$$\sum_{d_w\in D_w} y(w,d_w) = 1 \qquad \forall w \in N(v) \tag{8}$$

$$y(w,d_w) \in \{0,1\} \qquad \forall w \in N(v), d_w \in D_w \tag{9}$$

The constraints (8) and (9) enforce that for each neighbor of v exactly one frequency is chosen. For a given choice of frequencies $d^*_{N(v)}$ the right-hand sides of constraints (7) are the penalties incurred with each of the corresponding frequencies for v. Thus, a frequency d_v with smallest penalty determines the highest value z can obtain for the particular choice of frequencies for the neighbors of v. For each possible assignment of frequencies to the neighbors of v we determine this value. The worst choice of $d^*_{N(v)}$ is one for which this value is maximal. This choice determines the value of z, and so $u(v,\delta(v))$.

Frequencies $d_v \in D_v$ for which $q(v,d_v) > u(v,\delta(v))$ can be removed from the domain. In the preprocessing phase such frequencies are removed for all vertices. Also, partial feasible solutions $d_S \in D_S$, $v \in S$, for which the penalty incurred by the frequency assigned to v and its incident edges is higher than $u(v,\delta(v))$ need not be considered.

The above technique can be generalized to sets of vertices, instead of single vertices. Consider a set $S \subset V$ with its set of assignments D_S.

$$u(S,\delta(S)) = \max \quad z \tag{10}$$

$$\text{s.t.} \quad z \le q(S,d_S) + \sum_{w\in N(S)} \sum_{d_w\in D_w} \sum_{v\in N(w)\cap S} p(v,d_v,w,d_w) y(w,d_w)$$

$$\forall d_S \in D_S \tag{11}$$

$$\sum_{d_w\in D_w} y(w,d_w) = 1 \qquad \forall w \in N(S) \tag{12}$$

$$y(w,d_w) \in \{0,1\} \qquad \forall w \in N(S), d_w \in D_w \tag{13}$$

Here, $q(S, d_S)$ denote the total penalty involved by an assignment d_S, i.e.,

$$q(S, d_S) = \sum_{v \in S} q(v, d_v) + \sum_{\{v,w\} \in E[S]} p(v, d_v, w, d_w)$$

This value is a lower bound on the total penalty involved in any complete assignment based on the partial assignment d_S. So, if $q(S, d_S) > u(S, \delta(S))$, then this partial assignment cannot be extended to an optimal complete assignment. Hence, it can be removed from the set of assignments D_S. An even better lower bound on the penalty in any complete assignment is given by the total penalty incurred by the subgraph $G[S]$ and the edges $\delta(S)$, i.e.

$$l(S, \delta(S), d_S) = q(S, d_S) + \sum_{w \in N(S)} \min_{d_w \in D_w} \left\{ \sum_{v \in N(w) \cap S} p(v, d_v, w, d_w) \right\}$$

Whenever $l(S, \delta(S), d_S) > u(S, \delta(S))$, we can remove d_S from our set of assignments for S. We will call an assignment non-redundant if $l(S, \delta(S), d_S) \leq u(S, \delta(S))$.

The upper bound $u(S, \delta(S))$ is especially powerful if the number of edges in the cut-set $\delta(S,)$ is small, or if the sum of the maximum penalties incurred by the cut-set edges is not too large. If the upper bound $u(S, \delta(S)) = 0$ for a subset S, then we know that given any assignment to the vertices $V \setminus S$, the partial solution can be extended to a complete solution without additional penalty. This implies that we can remove the subset S and the edges $\delta(S)$ from the constraint graph.

A similar upper bounding technique can be applied to a small extension of the set S and the edges in its cut-set, i.e., an upper bound for the induced subgraph $G[S]$, the edges $\delta(S)$ and the vertices $N(S)$.

Note that, if $T \subseteq S$, $u(T, \delta(T)) \leq u(S, \delta(S))$, which implies that the upper bound for S is also valid for T. The upper bound $u(S, \delta(S))$ can also be used in combination with lower bounds. Let $S, T \subset V$ be disjoint subsets, and let $l(S)$ be a lower bound on the penalty incurred by $G[S]$. Then, an upper bound $u(T, \delta(T))$ is given by $u(T, \delta(T)) = u(T, \delta(T)) - l(S)$. Similarly, if $l(S, \delta(S))$ is a lower bound on the penalty incurred by $G[S]$ and the edges $\delta(S)$, then an upper bound for $G[T]$ is given by $u(T) = u(S \cup T, \delta(S \cup T)) - l(S, \delta(S))$.

The main problem with $u(S, \delta(S))$ is that it may take quite some time to compute it. It may be preferable to compute the value of some relaxation of (10)-(13). The LP-relaxation does not generate really powerful upper bounds. Our choice is therefore to relax (10)-(13) by taking a subset of the constraints (11), i.e., a number of partial feasible solutions with low $q(S, d_S)$. In case we restrict ourselves to one good partial solution d_S^* for S we can solve the relaxed problem by inspection, and use this as an upper estimate of $u(S, \delta(S))$:

$$\begin{aligned} u(S, \delta(S)) &\leq q(S, d_S^*) + \sum_{w \in N(S)} \sum_{d_w \in D_w} \sum_{v \in N(w) \cap S} p(v, d_v^*, w, d_w) y(w, d_w) \\ &= q(S, d_S^*) + \sum_{w \in N(S)} \max_{d_w \in D_w} \sum_{v \in N(w) \cap S} p(v, d_v^*, w, d_w) \end{aligned} \tag{14}$$

Note that good partial solutions are usually available through heuristics, or are generated in the dynamic programming algorithm.

6.3.2 Dominance

Upper bounding techniques are a quick way to eliminate the worst partial feasible solutions, but these techniques sometimes only remove a small fraction of the solutions that are redundant. In this subsection we develop techniques that remove partial solutions for which there exist better alternatives. Consider again the example of Figure 7. Though frequency 3 could not be removed from D_v using the upper bound, we can verify in Table 1 that for no choice of frequencies for the neighbors of v frequency 3 is the unique optimal choice. In other words, in any solution where frequency 3 is chosen we can replace it by another frequency without obtaining a worse solution. Therefore, we maintain at least one of the optimal solutions by removing this frequency from D_v.

The abstract concept of dominance is as follows. Let $v \in V$. Consider all partial solutions of $N(v)$. If these solutions can be extended with a frequency of $D_v \setminus \{d_v^*\}$ to solutions at minimum cost, then d_v^* is not necessary to obtain an optimal solution. Therefore d_v^* can be removed from D_v. We say that d_v^* is *dominated* by the frequencies in $D_v \setminus \{d_v^*\}$. This concept can also be generalized to sets of vertices, similar to the generalization of the upper bounds to sets $S \subset V$. Let d_S^* be an assignment to S, then d_S^* is dominated by the other non-redundant assignments $D_S \setminus \{d_S^*\}$ if every partial feasible solution of $N(S)$ can be extended to a solution at minimum cost with an assignment of $D_S \setminus \{d_S^*\}$.

To find out whether d_S^* is dominated by $D_S \setminus \{d_S^*\}$, we formulate the following feasibility problem, which has a feasible solution if and only if d_S^* is the unique minimum for some choice of frequencies of the neighbors. Therefore, it is dominated if and only if this problem has no solution. The binary variables $y(w, d_w)$ used in this formulation have the same meaning as in the previous subsection: $y(w, d_w) = 1$ if frequency d_w is chosen for node w, and 0 otherwise. Then the feasibility problem reads

$$q(S, d_S^*) + \sum_{w \in N(S)} \sum_{d_w \in D_w} \{ \sum_{v \in N(w) \cap S} p(v, d_v^*, w, d_w) \} y(w, d_w)$$

$$< q(S, d_S) + \sum_{w \in N(v)} \sum_{d_w \in D_w} \{ \sum_{v \in N(w) \cap S} p(v, d_v, w, d_w) \} y(w, d_w) \quad \forall d_S \in D_S \setminus \{d_S^*\} \tag{15}$$

$$\sum_{d_w \in D_w} y(w, d_w) = 1 \qquad \forall w \in N(S) \tag{16}$$

$$y(w, d_w) \in \{0, 1\} \qquad \forall w \in N(S) \forall d_w \in D_w \tag{17}$$

For any solution of $N(S)$ the constraints (15) state that the penalty of d_S^*, the left hand side (LHS), should be smaller than the penalty of each $d_S \in D_S \setminus \{d_S^*\}$, the right hand side (RHS). In other words, if there is a solution of $N(S)$ with this property, then d_S^* is the unique optimum for this solution, and thus it is *not* dominated by the other frequencies in $D_S \setminus \{d_S^*\}$.

To transform (15)-(17) into an integer linear program we introduce a variable z, which denotes the maximum difference between the RHS and LHS of (15), i.e., it is a measure of the "minimality" of d_S^*.

$$\max \quad z \tag{18}$$

$$\text{s.t.} \quad z \leq q(S, d_S) - q(S, d_S^*) + \sum_{w \in N(v)} \sum_{d_w \in D_w} \{ \sum_{v \in N(w) \cap S} \Delta p(v, d_v, d_v^*, w, d_w) \} y(w, d_w)$$

$$\forall d_S \in D_S \setminus \{d_S^*\} \tag{19}$$

$$\sum_{d_w \in D_w} y(w, d_w) = 1 \qquad \forall w \in N(S) \tag{20}$$

$$y(w, d_w) \in \{0, 1\} \tag{21}$$

where $\Delta p(v, d_v, d_v^*, w, d_w) = p(v, d_v, w, d_w) - p(v, d_v^*, w, d_w)$. Clearly, if $z > 0$, then d_S^* is not dominated, since the feasibility problem has a solution; otherwise, if $z \leq 0$, d_S^* is dominated.

The formulation (18)-(21) resembles the upper bound formulation (10)-(13). Moreover, this problem has to be solved for all non-redundant assignments. Therefore, we again relax the problem by removing constraints. For a good partial solution d_S we generate the corresponding constraint (19). This restricted problem can then be approximated by inspection. From (19) we get:

$$\begin{aligned} z &\leq q(S, d_S) - q(S, d_S^*) + \sum_{w \in N(v)} \sum_{d_w \in D_w} \{ \sum_{v \in N(w) \cap S} \Delta p(v, d_v, d_v^*, w, d_w) \} y(w, d_w) \\ &= q(S, d_S) - q(S, d_S^*) + \sum_{w \in N(v)} \max_{d_w \in D_w} \{ \sum_{v \in N(w) \cap S} \Delta p(v, d_v, d_v^*, w, d_w) \} \\ &\leq q(S, d_S) - q(S, d_S^*) + \sum_{\{v,w\} \in \delta(S)} \max_{d_w \in D_w} \Delta p(v, d_v, d_v^*, w, d_w) \end{aligned} \tag{22}$$

So, if the RHS of (22) is already ≤ 0, then d_S^* is dominated.

7 Iterative Version of the Dynamic Programming Algorithm

Both time and memory are insufficient to solve large instances with the dynamic programming algorithm described in Section 5, even if we use the reduction techniques of Section 6. During the algorithm, the number of non-redundant assignments explodes for these instances. We can point out two reasons. On the one hand, the width of our tree decomposition is too large. On the other hand, the number of frequencies available for a vertex is too large. In this section we focus on this last reason. Instead of assigning frequencies to the vertices, we propose to assign subsets of frequencies. So, we partition the domain of a vertex in a number of subsets, and assign

one of them to the vertex. To handle these subsets as frequencies of a new FAP, we have to harmonize the vertex and edge-penalties for all frequencies in a subset. We take as penalty the minimum of the individual penalties. In this way the solution value of the new FAP is a lower bound for the original problem. We can extend this idea to an iterative method which provides a sequence of lower bounds for the original instance. The dynamic programming algorithm is used as a subroutine to solve the FAPs with the substantially smaller domains. Contrary to the original FAP, time and memory are sufficient to solve these FAPs, because they are much smaller.

The idea of the method is that we identify a subset of the domain with each vertex. The vertex- and edge-penalties for these subsets are estimated from below. For example, consider the matrix of edge-penalties given in Figure 8(a). The level of interference on this edge is 10 if the difference between the frequencies is less than 2. If we divide the frequencies in two groups $\{1, 2\}$, and $\{3, 4\}$, we obtain 4 blocks in the table of edge-penalties with (almost) the same values. In most cases there is no difference between the penalties as long as the pairs of frequencies are in the same block. Therefore, let us construct a new FAP in which we have to assign either the subset $\{1, 2\}$ or the subset $\{3, 4\}$ to the vertices. The edge-penalties in this new FAP are given by the minimum of the values in each block (see Figure 8(b)). Solving this substantially smaller problem provides a lower bound for the optimal value of the original problem. The quality of the lower bound depends on the size of the blocks: many small blocks will provide a better lower bound than a small number of large blocks. In most real-life instances the block structure of the penalty matrices arises naturally, since the available frequencies for an antenna can be divided in groups of frequencies that are in the same part of the spectrum.

d_v, d_w	1	2	3	4
1	10	10	0	0
2	10	10	10	0
3	0	10	10	10
4	0	0	10	10

(a) original penalty matrix

d_v, d_w	$\{1, 2\}$	$\{3, 4\}$
$\{1, 2\}$	10	0
$\{3, 4\}$	0	10

(b) new penalty matrix

Figure 8: Example to illustrate the idea behind the iterative algorithm

This idea can be formalized in the following algorithm. We start with the original problem $P = (G, D, p, q)$ and we partition for every vertex $v \in V$ the domain D_v in an initial number of n_v subsets $D_v^1, \ldots, D_v^{n_v}$. This partition is, for example, based on a natural partition of the frequencies in groups of frequencies that are in the same part of the spectrum.

Next, we construct a new FAP $P' = (G, D', p', q')$, with

- domains $D'_v = \{1, \ldots, n_v\}$ for all vertices $v \in V$,
- vertex-penalties $q'(v, i) = \min_{d_v \in D_v^i} q(v, d_v)$ for every vertex $v \in V$, $i \in D'_v$, and

- edge-penalties $p'(v,i,w,j) = \min_{d_v \in D_v^i} \min_{d_w \in D_w^j} p(v,d_v,w,d_w)$ for every edge $\{v,w\} \in E$, $i \in D'_v$, $j \in D'_w$.

So, P' is defined on the same graph as P, and the domains of P' correspond with the subsets D_v^i, $i = 1,\ldots,n_v$. Since the vertex and edge-penalties in P' are the minimum of the penalties in the corresponding subset(s), the optimal value of the problem P' provides a lower bound for the optimal value of the original problem P.

Our way to obtain a sequence of non-decreasing lower bounds is based on an iterative refinement of the domain-subsets. A partition $\tilde{D}_v^1,\ldots,\tilde{D}_v^m$ of a domain D_v is called a refinement of another partition $\bar{D}_v^1,\ldots,\bar{D}_v^n$, if for every subset $\tilde{D}_v^i$, $i = 1,\ldots,m$, there exists a subset $\bar{D}_v^j$, $j \in \{1,\ldots,n\}$ in the second partition for which $D_v^i \subseteq D_v^j$. If $\tilde{P}$ and $\bar{P}$ are FAPs corresponding to these partitions, then the value of the optimal solution of $\tilde{P}$ will be at least as high as the value of the optimal solution of $\bar{P}$, which implies that $\tilde{P}$ provides a lower bound that is greater than or equal to the lower bound provided by $\bar{P}$.

Now, we can extend the algorithm to obtain a lower bound to an algorithm that provides a sequence of non-decreasing lower bounds as follows. We construct a problem P' which provides us with the first lower bound. Next, we refine the partition of the subsets, and again construct a FAP P' which hopefully provides us with a better lower bound. We can repeat the refinement of the partition as long as the efforts to solve the problem P' is reasonable in both time and memory. A flowchart of this algorithm is presented in Figure 9.

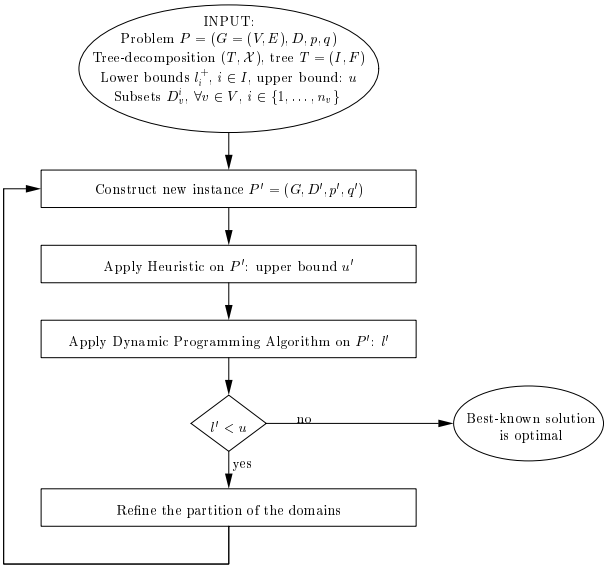
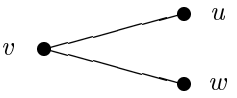


Figure 9: Iterative version of the algorithm

Whatever refinement procedure (i.e., for which vertices do we refine the partition, and how do we refine the partition) we apply, a guarantee that the new lower bound will be strictly greater than the old lower bound cannot be given in general. However, if for all vertices $v \in V$, the

domain-subset that corresponds to the optimal solution of P' is not partitioned in the refinement procedure, then the 'old' optimal solution is still optimal in the new problem P'. This implies that a refinement can only be effective if at least one selected domain-subset is refined. Therefore, for each refinement we select one vertex v, for which we partition the assigned subset. To speed up the process in practice, we do not apply the dynamic programming algorithm after each single refinement, but after the refinement of the domains for a subset of the vertices $S \subseteq V$.

For a partition of the assigned subset for a vertex $v \in V$ we can compute an upper bound on the increase of the value of P'. This upper bound is used as criteria to select a partition. Consider



penalties		D'_u	D'_w	total
D'_v	d_v^1	1	0	1
	d_v^2	0	1	1
minimum		0	0	1

Figure 10: Example to illustrate the partition of assigned subsets

the example of Figure 10. Let $D'_v = \{d_v^1, d_v^2\}$ be the assigned subset to v, and let D'_u and D'_w be the assigned subsets to the neighbors u and w, respectively. The total penalty incurred by this assignment is 0. However, if we either assign d_v^1 or d_v^2 to v, then the total penalty will be one. Hence, partition of the subset may lead to an increase of the value of P'. It cannot be guaranteed however, since the new optimal assigned may select a subset other than $\{d_v^1\}$ or $\{d_v^2\}$.

In general, an upper bound on the increase of the optimal value by a partition of the assigned subset can be computed as follows. We restrict ourselves to a partition of the assigned domain-subset in two domain-subsets, but the procedure can easily be extended to a partition in more than two domain-subsets. The procedure can also be generalized to subsets of vertices instead of single vertices. Let $v \in V$, and D'_v be the domain-subset that corresponds to the optimal assignment. If we partition D'_v in A_v and $D'_v \setminus A_v$, then the value of the problem P' will increase with at most $\Delta\pi(v, A_v)$,

$$\Delta\pi(v, A_v) = \min\{\pi(v, A_v), \pi(v, D'_v \setminus A_v)\} - \pi(v, D'_v)$$

where

$$\pi(v, D) = \min_{d_v \in D} q(v, d_v) + \sum_{w \in N(v)} \min_{d_v \in D} \min_{d_w \in D'_w} p(v, d_v, w, d_w)$$

Among all partitions A_v, $D'_v \setminus A_v$, the best partition, according to the value $\Delta\pi(v, A_v)$, is $A_v^* = \arg\max_{A_v \subset D'_v} \Delta\pi(v, A_v)$. If $\Delta\pi(v, A_v^*) = 0$ then no single refinement of the partition for vertex v will result in an increase of the lower bound for P. Therefore, the subset S for which we will partition the assigned subset is given by the vertices for which $\Delta\pi(v, A_v^*) > 0$.

The iterative method can be separated from the dynamic programming algorithm. In principle we can use any exact algorithm to solve the consecutive FAPs. However, the use of the dynamic

programming algorithm of Section 5 enables us to use information of previous solved problems. More precisely, during the computation of the optimal solution of a previous problem P', we obtain for all $i \in I$ a lower bound $l(Y_i, \delta(Y_i))$ for the penalty incurred by $G[Y_i]$ and the edges $\delta(Y_i)$. These values are also lower bounds on the penalty in the new problem P', which implies that we can compute upper bounds $u(V \setminus Y_i) = u' - l(Y_i, \delta(Y_i))$ for all $i \in I$. Here, u' is a general upper bound for the new problem P' which can be computed by one of the heuristics available for the FAP. If the increase of u' is not too large for two consecutive problems P', then the upper bounds for the subsets are often relatively strong.

8 Computational Results

In this section we report on the results we have obtained using the approach described in the previous sections. We tested the methods described in this paper on real-life instances obtained from the CALMA-project [19]. The set of instances consists of two parts. The CELAR instances are real-life problems from a military application. The GRAPH instances are randomly generated problems with the same characteristics. We only used the 11 so-called penalty-instances, since for the other instances the objective is either to minimize the frequency span, or the minimize the number of frequencies used. In this section we solve 7 out of the 11 instances to optimality and we obtain good lower bounds for the other instances. Before this study non-trivial lower bounds were only available for 2 instances.

The solution procedure can be divided in four parts, each of which is analyzed in the forthcoming subsections. In Section 8.1 we report on the results obtained with the preprocessing techniques of Section 6. The results of the heuristic to construct a tree decomposition of Section 4 are presented in Section 8.2. In Section 8.3, we show that some of the instances of the CALMA-project can be solved to optimality with the dynamic programming algorithm of Section 5. Furthermore, we compare the performance of the dynamic programming algorithm with the polyhedral approach on 5 small test instances that have been constructed from one of the CELAR instances. Section 8.4 is devoted to the lower bounds which were obtained with the iterative version of the algorithm described in Section 7.

All implementations have been carried out in C++. The programs for the dynamic programming algorithm and the iterative version of the algorithm were running on a DEC 2100 A500MP workstation with 128Mb internal memory. The programs for preprocessing, for the construction of a tree decomposition, and for the computation of upper bounds for single vertices were executed on a Pentium II - 233 Mhz Personal Computer with 32Mb internal memory. We used the callable library of CPLEX 4.0 to solve (integer) linear programming problems.

8.1 Preprocessing

We start our computations with the application of the graph and domain reduction techniques described in Section 6. The following procedure is repeated as long as the size of the problem is reduced. First of all, we apply penalty shifting from edges to vertices and from vertices to

the objective. Next, we apply the graph reduction techniques: the removal of vertices with only one domain element, the removal of edges with only zero penalty, and the removal of vertices of degree less than or equal to two. Then, we calculate the upper bound (14) for every vertex, and we apply the dominance test (22) for single vertices. If a frequency is dominated, then we remove this frequency from the domain. The dominance test (18)-(21) with $S = \{v\}$ yields no additional reduction. If, due to the upper bound and dominance test, a vertex with only one frequency occurs, we remove the vertex. We apply the same upper bound (14) and dominance test (22) for adjacent vertices. Contrary to the dominance test for a single vertex we cannot remove the frequencies of the combination in case it is dominated. Therefore, we increase the edge-penalty of this combination with an amount that guarantees that it will never occur in a non-redundant assignment. Moreover, if given a frequency $d_v \in D_v$, the combination (d_v, d_w) is dominated for all $d_w \in D_w$, we can remove the frequency d_v from the domain D_v.

	before preprocessing			after preprocessing				best	previous
instance	$\|V\|$	$\|E\|$	$\|D\|$	$\|V\|$	$\|E\|$	$\|D\|$	fixed	value	lower bound
CELAR 06	100	350	39.9	82	327	39.9	0	3389	0
CELAR 07	200	817	39.9	162	764	34.6	0	343592	0
CELAR 08	458	1655	39.5	365	1539	39.4	0	262	0
CELAR 09	340	1130	39.5	67	165	35.6	11391	15571	14969
CELAR 10	340	1130	39.5	0	0	-	31516	31516	31204
GRAPH 05	100	416	37.1	0	0	-	221	221	-
GRAPH 06	200	843	37.7	119	348	16.2	4112	4123	-
GRAPH 07	200	843	36.7	0	0	-	4324	4324	-
GRAPH 11	340	1425	37.7	340	1425	32.6	2553	3080	-
GRAPH 12	340	1255	37.6	61	123	15.3	11496	11827	-
GRAPH 13	458	1877	38.4	456	1874	38.1	8676	10110	-

Table 2: Statistics and preprocessing penalty-instances CALMA-project

In the Table 2 statistics for all penalty-instances before and after preprocessing are reported. Consecutively, we report the number of vertices ($|V|$) and the number of edges ($|E|$) in the constraint graph, and the average number of domain elements ($|D|$). In addition, we report the value that is fixed by the preprocessing phase, the best known value (see Kolen [14]), and the best known lower bound (cf. Hurkens and Tiourine [11]). For the GRAPH instances this lower bound is not available. Table 2 shows that 3 out of the 11 penalty-instances are solved by preprocessing only. For the instances CELAR 10 and GRAPH 07 this is mainly due to the vertex penalties, that cause the removal of many frequencies. The graph reduction in the instances CELAR 09 and GRAPH 12 can be explained in the same way. Table 2 also shows that there is a major difference between the real-life CELAR instances and the randomly generated GRAPH instances. The fixed value for the CELAR instances without vertex-penalties is simply zero, whereas for the GRAPH instances 80% or more of the best known value can be fixed. This difference can be explained by the effectiveness of the different preprocessing rules. For the CELAR instances, the main part of the reduction is due to the removal of vertices with degree less than or equal to two, whereas the main part of the reduction for the GRAPH instances is due to penalty shifting (fixing) and the dominance test (22) for single vertices. In fact, for the

instance GRAPH 05, a first round of shifting penalties resulted in a lower bound of 220. As a consequence, many domain elements could be removed from the problem, and the constraint graph reduced substantially. A new round of shifting penalties resulted in the proof of optimality of the best known solution. The running time of the preprocessing phase is within a minute for all penalty-instances.

8.2 Construction of Tree-decompositions

The second step in solving a FAP is the construction of a tree decomposition of the preprocessed constraint graph.

instance	$\|V\|$	$\|E\|$	width	max clique - 1	cpu-time (sec)
CELAR 06	82	327	11	10	17
CELAR 07	162	764	17	10	176
CELAR 08	365	1539	18	10	802
CELAR 09	67	165	7	7	0
GRAPH 06	119	348	17	5	137
GRAPH 11	340	1425	104	7	19749
GRAPH 12	61	123	4	4	6
GRAPH 13	456	1874	133	6	63586

Table 3: Construction of a tree decomposition

In Table 3 we report on the results of the heuristic of Section 4. We also report the maximum clique size minus one. Since every clique should be in at least one node of the tree, this value is a lower bound for the treewidth of a graph. Table 3 shows that the gap between the width and the lower bound varies from zero for small instances to very large for the large GRAPH instances. For these instances it is not clear which bound is poor. We have tried several variants of our heuristic to improve the width of the tree decomposition, but without any success.

8.3 Dynamic Programming Algorithm

In this subsection we report the computational results obtained with the dynamic programming algorithm of Section 5. The order in which we calculate all non-redundant assignments for the subsets Y_i, $i \in I$ is based on the available upper bounds (14) for $S = Y_i$. The sets Y_i are ordered according to non-decreasing $u(Y_i, \delta(Y_i))$. During the dynamic programming algorithm the upper bounds for the subsets are updated every time we obtain a new lower bound for a subset by computing all non-redundant assignments. If an upper bound for a subset decreases and all non-redundant assignments are already computed, we remove all assignments with penalty larger than the new upper bound. The dynamic programming algorithm is used with and without applying a dominance test for the subsets Y_i. As dominance test, we solve the linear programming relaxation of (18)-(21) with a limited number of constraints (19).

A first test of the dynamic programming algorithm is performed on 5 instances with size of the domains between 2 and 6 for all vertices. These instances were constructed from the instance CELAR 6 by taking a subset of the domain elements of predefined size. In Koster, van Hoesel and Kolen [15] the polyhedral approach is tested on these instances. The tree decomposition approach is tested on these instances with and without using the dominance test (18)-(21). In

instance	$\|D_v\|$	CPU-time for		
		polyhedral method	DP without dominance	DP with dominance
CELAR6a	2	3.5	3.8	8.8
CELAR6b	3	84.1	38.4	35.1
CELAR6c	4	11785.5	408.6	219.2
CELAR6d	5	22501.2	2306.5	592.9
CELAR6e	6	75570.5	-	2399.4

Table 4: Computational results dynamic programming algorithm test instances

Table 4 the computation times of the polyhedral method and the tree decomposition approach are compared. Without using dominance the dynamic programming algorithm cannot solve the largest instance. At some point during the dynamic programming algorithm the number of non-redundant assignments for a subset is too large to store into the memory of our computer. The table shows that both dynamic programming algorithms are competitive or substantially faster than the polyhedral method. We also may conclude that the use of the dominance test in the dynamic programming algorithm speeds up the process for these instances.

The dynamic programming algorithm is also performed on the original penalty-instances. The polyhedral method is not able to solve these instances, or even to generate non-trivial lower bounds. Table 5 shows the results that are obtained with the dynamic programming algorithm

instance	optimal value	CPU-time (sec)
CELAR 06	3389	27102
CELAR 07	-	-
CELAR 08	-	-
CELAR 09	15571	23
GRAPH 06	4123	29
GRAPH 11	-	-
GRAPH 12	11827	11
GRAPH 13	-	-

Table 5: Computational results dynamic programming algorithm

without dominance. Experiments with the dominance test did not result in a better performance of the algorithm for these instances. The instances CELAR 09, GRAPH 06 and GRAPH 12 can be solved very efficiently with this method. After more than 7.5 hours the algorithm was able to prove that the best known solution was optimal for this instance as well. Figure 11 shows the number of non-redundant assignments during the process compared with the theoretical number.

The optimal value for all these instances is equal to the best known. The instance `CELAR 06` is more difficult to solve. Mainly due to limitations in computer memory, we are not able to solve the other instances.

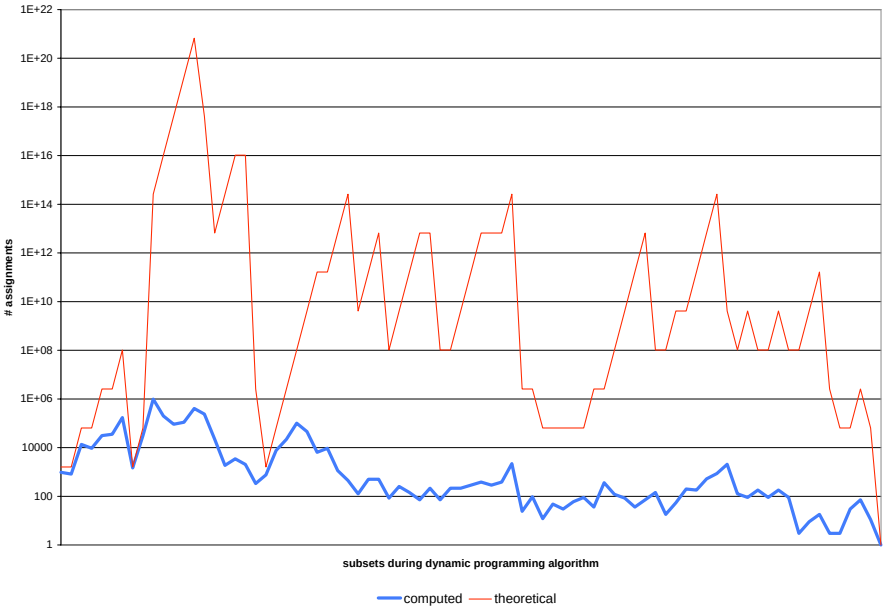


Figure 11: Number of non-redundant assignments CELAR 06

8.4 Iterative version

Table 5 in the previous subsection shows that the dynamic programming algorithm is not able to solve several instances. For these problems we apply the iterative version of the algorithm of Section 7. Before we start our computations we have to partition all domains in an initial number of subsets. In our experiments we start with either 2 or 4 subsets for every vertex. The partition of the subsets is based on a natural partition of the frequencies in the radio spectrum.

In each iteration of the algorithm, first a heuristic is applied to obtain an upper bound for the instance. In our computational experiments we used the genetic algorithm developed by Kolen [14]. Then, we apply the dynamic programming algorithm in the same way as in the previous subsection. As last step in an iteration, we partition the selected domain-subset of the vertices in the set S. As described in Section 7, we base our selection of S and the actual partitions on the values $\Delta\pi(v, A_v^*)$. We limit the set S to at most 20 vertices. Moreover, we do not compute $\Delta\pi(v, A_v)$ for every partition of the selected domain-subset D'_v, but only for partitions of the form $\{1, \ldots, i\}, \{i+1, \ldots, |D'_v|\}$.

In Table 6 we report the results we obtained in this way for the instances that we could not solve with the original dynamic programming algorithm. For `CELAR 07` we obtain a lower bound

instance	initial # subsets	lower bound after preprocessing	lower bound iterative algorithm	upper bound	CPU-time (sec)
CELAR 06	2	0	3388	3389	13734
	4		3388		9429
CELAR 07	2	0	243066	343592	259022
	4		300000		275736
CELAR 08	2	0	87	262	313168
	4		74		12482
GRAPH 11	4	2553	-	3080	-
GRAPH 13	4	8676	-	10110	-

Table 6: Computational results iterative version of the algorithm

that is within 12.5% of the best known value. Both for CELAR 07 and CELAR 08 the values are the first non-trivial lower bounds. For the instances GRAPH 11 and GRAPH 13, the width of the tree decomposition is too large to apply the dynamic programming algorithm with any success. We also apply the iterative version to the instance CELAR 06. If we either start with an initial number of subsets of 2 or 4, we obtain a lower bound that is one away from optimal in third the time (or half the time) that is needed to solve the problem to optimality with the original dynamic programming algorithm. Figure 12 shows the improvement of the lower bound during

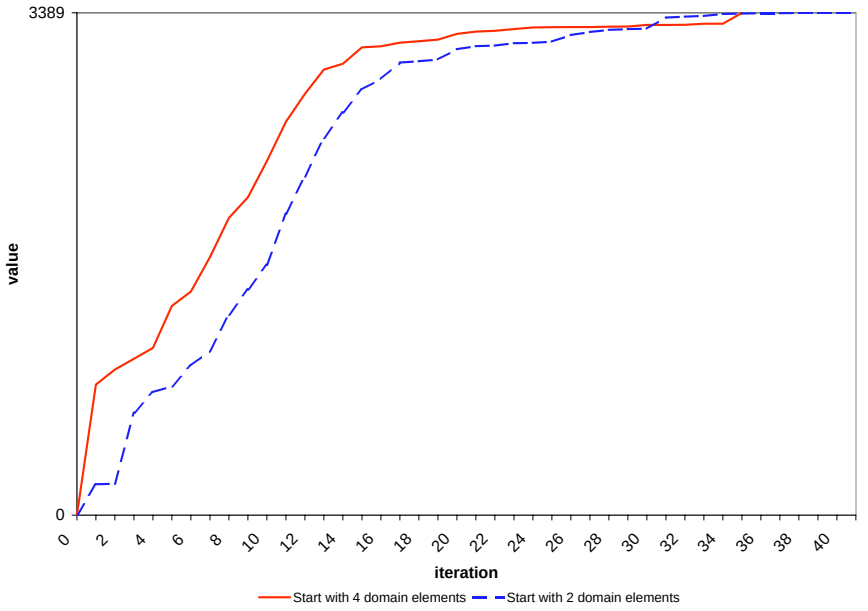


Figure 12: Lower bounds CELAR 06

the process. In case we start with 2 subsets per vertex we need 38 iterations to achieve the lower bound of 3388, whereas if we start with 4 subsets per vertex we need 35 iterations. Figure 13 shows a histogram with the vertices as a function of the number of subsets after the last iteration.

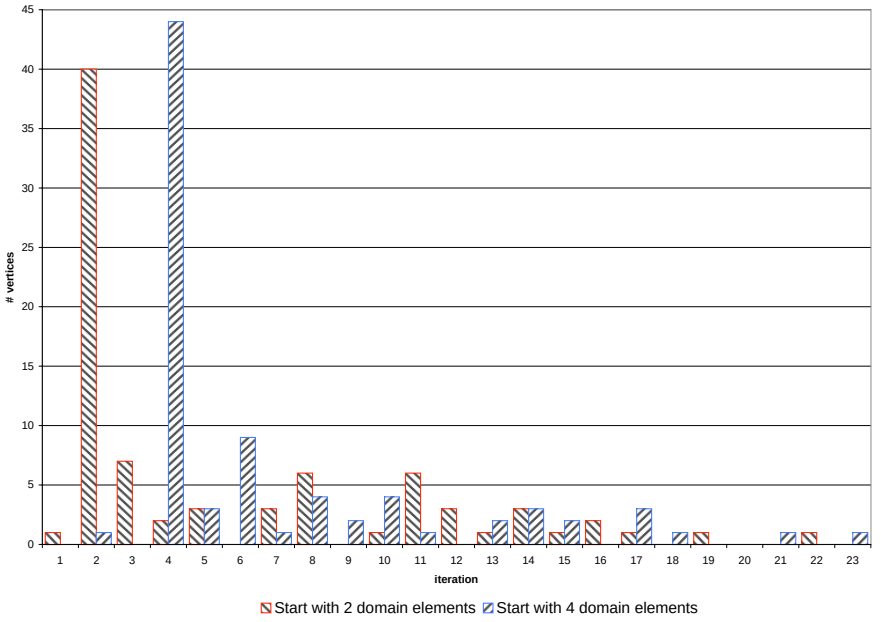


Figure 13: Number of vertices as function of number of domain-subsets for CELAR 06 at the end of the iterative algorithm.

It shows that only for a restricted number of vertices the domain is refined during the process.

9 Concluding Remarks

In this paper we described a computational study to use the concept of tree decomposition to solve frequency assignment problems to optimality. We showed that the method, although theoretically polynomial in both time and space requirements, can only be applied to real-life problem instances if we use additional reduction techniques, like graph reduction, upper bounding and dominance. Even with these techniques, it is not sure that the instances can be solved. Therefore, we presented an iterative version of the algorithm which can be used to obtain lower bounds for most of the instances. For a set of real-life instances, we proved optimality for several instances, whereas we obtained the first non-trivial lower bounds for the other instances. Other methods, like integer programming techniques were not able to solve these instances.

Based on these results, we state two directions for further research. One way is to embed either the dynamic programming algorithm or the iterative algorithm in a branch-and-bound framework. Hopefully, this result in even better lower bounds. Another way for further research is the application of this method to other hard combinatorial optimization problems. It is worthwhile to investigate the possibilities of this method for problems that are based on a graph, and which cannot be solved by the current solution methods.

Acknowledgement

The authors would like to thank Rudolf Müller for his suggestions which resulted in the iterative algorithm of Section 7.

References

[1] K.I. Aardal, A. Hipolito, C.P.M. van Hoesel, and B. Janssen. "A branch-and-cut algorithm for the frequency assignment problem.". Research Memorandum 96/011, Maastricht University, 1996.

[2] K.I. Aardal, C.P.M. Van Hoesel, A.M.C.A. Koster, C. Mannino, and A. Sassano. Models and solutions techniques for the frequency assignment problem. Working paper, Maastricht University, 1999.

[3] S. Arnborg, D.G. Corneil, and A. Proskurowski. Complexity of finding embeddins in a k-tree. *SIAM Journal on Algebraic and Discrete Methods*, 8:277–284, 1987.

[4] H.L. Bodlaender. "A tourist guide through treewidth". *Acta Cybernetica*, 11(1-2):1–21, 1993.

[5] H.L. Bodlaender. "A linear time algorithm for finding tree-decompositions of small treewidth". *SIAM Journal on Computing*, 25(6):1305–1317, 1996.

[6] R. Borndörfer, A. Eisenblätter, M. Grötschel, and A. Martin. Frequency assignment in cellular phone networks. *Annals of Operations Research*, 76:73–94, 1998.

[7] D.J. Castelino, S. Hurley, and N.M. Stephens. A tabu search algorithm for frequency assignment. *Annals of Operations Research*, 10:301–319, 1996.

[8] M. R. Garey and D.S. Johnson. *"Computers and intractability: a guide to the Theory of NP-Completeness"*. Freeman and Company, N.Y., 1979.

[9] R.E. Gomory and T.C. Hu. Multi-terminal network flows. *Journal of SIAM*, 9:551–570, 1961.

[10] W.K. Hale. Frequency assignment: Theory and applications. *Proceedings of the IEEE*, 68:1497–1514, 1980.

[11] C.A.J. Hurkens and S.R. Tiourine. Upper and lower bounding techniques for frequency assignment problems. Technical Report COSOR 95-34, Eindhoven University of Technology, 1995.

[12] J. Janssen and K. Kilakos. Polyhedral analysis of channel assignment problems: (i) tours. Technical Report CDAM-96-17, London School of Economics, 1996.

[13] B. Jaumard, O. Marcotte, C. Meyer, and T. Vovor. Comparison of column generation models for channel assignment in cellular networks. *Discrete Applied Mathematics*, October 1999.

[14] A.W.J. Kolen. A genetic algorithm for the partial binary constraint satisfaction problem: An application to a frequency assignment problem. Technical report, Maastricht University, 1999.

[15] A.M.C.A. Koster, C.P.M. Van Hoesel, and A.W.J. Kolen. The partial constraint satisfaction problem: Facets and lifting theorems. *Operations Research Letters*, 23(3-5):89–97, 1998.

[16] K. Menger. Zur allgemeinen kurventheorie. *Fundamenta Mathematicae*, 10:96–115, 1927.

[17] N. Robertson and P.D. Seymour. Graph minors. I. excluding a forest. *Journal of Combinatorial Theory, Series B*, 35:39–61, 1983.

[18] N. Robertson and P.D. Seymour. Graph minors. II. algorithmic aspects of tree-width. *Journal of Algorithms*, 7:309–322, 1986.

[19] S. Tiourine, C. Hurkens, and J.K. Lenstra. "An overview of algorithmic approaches to frequency assignment problems". In *Calma Symposium on Combinatorial Algorithms for Military Applications*, pages 53–62, 1995. Available at http://www.win.tue.nl/math/bs/comb_opt/hurkens/calma.html.