Frequency assignment problems: Subgraph generation for lower bounds

Stuart Allen

smallen@glam.ac.uk

Division of Mathematics and Computing

University of Glamorgan

Pontypridd CF37 1DL

Contents

Abstract

To generate good lower bounds for minimum span frequency assignment problems requires the identification of critical subproblems. A clique in the constraint graph often leads to good bounds, however for some problems the clique must be modified to improve the bound. This can be time consuming, requires manual intervention and is dependent on the initial clique obtained and the specific problem. For practical use, a simple bounding technique is needed that can be universally applied to all problems without modification. Two algorithms based on metaheuristics are presented in this report that aim to find a subproblem with the best lower bound possible. This avoids the need for clique detection routines and manual intervention, and gives a robust and automatic method for generating lower bounds for all minimum span problems. These algorithms use recent advances in generating lower bounds from mathematical programming formulations of the frequency assignment problem. Extensive testing on a wide range of problems shows that bounds from the new algorithms match those using previous techniques, and in some cases are significantly better.

Introduction

Given a radio network, the minimum span frequency assignment problem is to assign frequencies to the transmitters of the network, so that

- a given level of service quality is met at all possible reception points.
- the range of spectrum, or *span*, used is minimised.

This is a computationally hard problem. For networks with more than about 30 transmitters, exact algorithms take a prohibitive amount of time. Metaheuristics have been used successfully to generate good quality frequency assignments [2,6]. Although a metaheuristic algorithm cannot be guaranteed to find a best possible solution, it will ideally find a near optimal solution quickly. Therefore, in order to assess the quality of individual assignments, and to effectively compare different algorithms, it is important to be able to generate good quality lower bounds for the span of an assignment. The lower bounding techniques described in [1,3,8,16,17] have been found to be good, giving tight results in many cases. However, they depend on being able to identify a critical subproblem in the network that determines the span of the full network. This report demonstrates how the bounding techniques themselves can be used to generate these subproblems. A refinement of the methods for cellular problems is also described.

Section 1 – Lower Bounding Techniques

Graph theoretic representation of the Frequency Assignment Problem

The minimum span frequency assignment problem is commonly modelled using constraints concerning the frequencies that can be allocated to pairs of transmitters (referred to as binary constraints). This leads to a graph theoretic representation of the problem.

Definition A *constraint graph*[1] is a complete graph with vertices corresponding to the transmitters of a network. Each edge, ij, of the graph is labelled with an integer c_{ij} that gives a frequency separation requirement between the corresponding transmitters.

Definition A *frequency assignment* (or *channel assignment*) in a constraint graph G is a mapping $f: V(G) \to \{0,1,2,\ldots,K\}$ such that

$$|f(v_i) - f(v_j)| \geq c_{ij}$$

for all $v_i, v_j \in V(G)$. Such an assignment is also referred to as a *zero-violation assignment*. If some of the constraints are violated then f is an assignment with *constraint* violations. The value of K is the *span* of the assignment.

Definition If K is a minimum over all zero-violation assignments then the assignment is a *minimal assignment*. The minimal value of K is the *minimum span* of G, denoted $\mathrm{sp}(G)$.

Lower bounding techniques for minimum span frequency assignment

For a full description of lower bounding techniques for minimum span frequency assignment see [1,16,17]. This report will only make use of the most successful bounds.

---

[1] This definition of a constraint graph is equivalent to the definition of the modified constraint graph G' in [1,16,17].

Bounds from the Travelling Salesman Problem

It was noted in [13] that Hamiltonian paths[2] in a constraint graph are related to minimum span frequency assignments. In particular, any zero-violation frequency assignment can be used to construct a Hamiltonian path in G whose cost is at least that of the minimum such path. This gives the following result.

Theorem If G is a constraint graph, and $H(G)$ is the value of a minimum weight Hamiltonian path in G, then

$$\mathrm{sp}(G) \geq H(G).$$

Proof See [17]

Finding $H(G)$ is also known as the Open Symmetric Travelling Salesman Problem (TSP) and so this bound is referred to as the *TSP bound*.

Any Hamiltonian path in a constraint graph can be used to construct a frequency assignment as follows:

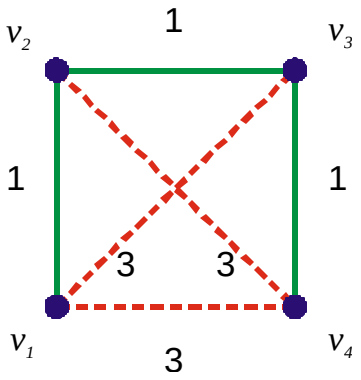
- Relabel the vertices so that $\{v_1,\ldots,v_{|V(G)|}\}$ is the Hamiltonian path of minimum weight.
- Let

$$f(v_1) = f_0$$

$$f(v_i) = f(v_{i-1}) + c_{ii-1} \qquad \text{for } i = 2,\ldots,|V(G)|$$

However, this assignment may contain constraint violations between vertices that are not consecutive in the path. For example, consider the path shown below:

[2] A Hamiltonian path in a graph is one that includes all vertices of the graph exactly once.



Consider the minimum weight Hamiltonian path v_1, v_2, v_3, v_4. If a frequency assignment is constructed as described above, then the frequencies assigned to v_1 and v_3 will differ by 2, violating the constraint between them. Similarly for vertices v_2 and v_4.

transmitter	v_1	v_2	v_3	v_4
frequency	0	1	2	3

Although the TSP bound can be used successfully for many frequency assignment problems [14,16], it has two drawbacks that prevent it from generating tight bounds in all situations.

1. **Calculation.** Finding a minimum weight Hamiltonian path is a NP-complete problem and so it is not always possible to find a solution in reasonable time.
2. **Tightness.** Since the TSP bound ignores frequency constraints between non-consecutive vertices, a frequency assignment constructed from a minimum weight Hamiltonian path is likely to contain constraint violations as indicated above. These ignored constraints mean that for some problems the TSP bound is not tight.

Bounds using mathematical programming

The drawbacks of the TSP bound can be addressed by considering an integer programming formulation of the bound. The graph G_0 is constructed from G by adding a dummy vertex v_0 joined by a single edge of weight 0 to all other vertices of G. A Hamiltonian circuit in G_0 is then equivalent to a Hamiltonian path in G. The variables x_{ij} specify the edges contained in the minimum weight Hamiltonian path:

$x_{ij} = 0$ if edge ij is not in the path

$x_{ij} = 1$ if edge ij is in the path.

The TSP bound can be formulated as an integer program as follows:

Minimise:	$\sum_{ij \in E(G_0)} c_{ij} x_{ij}$		
Subject to:	$\sum_{j:ij \in E(G_0)} x_{ij} = 2$	$i \in V(G_0)$	Degree constraints

	$\sum_{i \in S, j \in V \setminus S} x_{ij}$	$\geq$ 2	$S \subset V(G_0)$	Subtour elimination constraints
	x_{ij}	$\in$ {0,1}	$ij \in E(G_0)$	Integrality constraints

TSP IP

The problems with the TSP bound described in the previous section can be addressed as follows:

1. **Calculation.** Relaxations of the TSP can be used to generate bounds more quickly. Possible relaxations include the linear programming relaxation[3], the minimum spanning tree problem[4] and the perfect two matching problem. Polynomial time algorithms are available for all of these relaxations. However, by considering relaxations the bounds obtained may be worse.
2. **Tightness.** Extra constraints can be added to the integer program to take account of the frequency assignment constraints ignored by the original formulation. Although this may improve the bound obtained where possible, the time taken to obtain solutions may be significantly increased.

The Perfect Two-Matching Problem (PTMP) is a relaxation obtained from the TSP integer program by removing the subtour elimination constraints. In [16] it is shown that a solution to the PTMP is a good approximation to the TSP bound for frequency assignment problems. The PTMP has the advantage of being easier to solve; a polynomial time algorithm exists [11]. It is also guaranteed that solutions to the linear programming relaxation will have all variables taking half-integral values (i.e. each x_{ij} takes the value of either 0, ½ or 1).

Minimise: $\sum_{ij \in E(G_0)} c_{ij} x_{ij}$

[3] Obtained by replacing the integrality constraints by $x_{ij} \in [0,1]$.

[4] The use of the minimum spanning tree problem is described in [1].

Subject to:	$\sum_{j:ij \in E(G_0)} x_{ij}$	= 2	$i \in V(G_0)$	Degree constraints
	x_{ij}	$\in$ {0,1}	$ij \in E(G_0)$	Integrality constraints

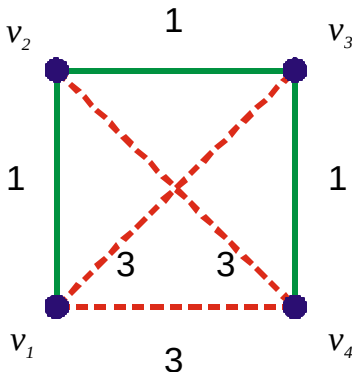
PTMP IP

In [16] extra constraints are described that improve the TSP bound by considering the frequency separation constraints between non-consecutive vertices on the Hamiltonian path. These constraints are based on additional *excess* variables for each edge of the path. An assignment can then be constructed from the path using both the edge weights and the excess variables to allow constraints to be satisfied with some slack. Again, the vertices are relabelled so that $\{v_1,\ldots,v_{|V(G)|}\}$ is the Hamiltonian path of minimum cost. Then let

$$f(v_1) = f_0$$

$$f(v_i) = f(v_{i-1}) + c_{ii-1} + e_{ii-1} \qquad \text{for } i = 2,\ldots,|V(G)|$$

where e_{ij} is the value of the excess variable for edge ij. Constraints between non-consecutive vertices can now be satisfied as shown below:



Consider the minimum weight Hamiltonian path v_1 , v_2 , v_3 , v_4. If a frequency assignment is constructed without excess values, then the frequencies assigned to v_1 and v_3 will differ by 2, violating the constraint between them. Similarly for vertices v_2 and v_4. Assigning an excess value of 1 to edge v_2v_3 allows all constraints to be satisfied.

transmitter	v_1	v_2	v_3	v_4
frequency	0	1	3	4

Constraints involving the excess variables can be formulated by considering each path P in the constraint graph that satisfies:

$$d(P) = c_{i_1 i_k} - (c_{i_1 i_2} + c_{i_2 i_3} + \cdots + c_{i_{k-1} i_k}) > 0$$

where $P = \{v_{i_1},\ldots,v_{i_k}\}$. $d(P)$ is called the *deficit* of the path and gives the amount by which the constraint between the end-vertices would fail if all other

constraints were met exactly. Let $\boldsymbol{P}$(G) denote the set of all paths P with positive deficit ($d(P) > 0$) and let $\boldsymbol{P}_k(G)$ denote the subset of $\boldsymbol{P}(G)$ consisting of all paths with length[5] k. The constraint for the path P with edge set $E(P)$ is then:

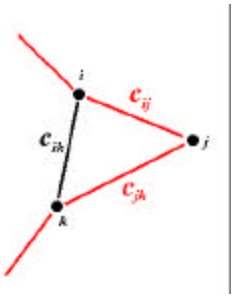
$$E_P \geq d(P)\left(X_P - \left(|E(P)| - 1\right)\right)$$

where $E_P = e_{i_1 i_2} + \cdots e_{i_{k-1} i_k}$ and $X_P = x_{i_1 i_2} + \cdots x_{i_{k-1} i_k}$. These constraints are referred to as *FAP constraints*. If all of the edges of P are chosen, $X_P = |E(P)|$ and the constraint becomes

$$E_P \geq d(P),$$

that is, enough excess must be added to overcome the deficit on the path. Otherwise E_P is unconstrained.

If a constraint is added for to the TSP integer program for all paths in $\boldsymbol{P}(G)$, a complete mathematical programming formulation of the minimum span frequency assignment problem is obtained. However, generally it is only practical to obtain solutions when a subset of the FAP constraints are used[6]. Solutions can be obtained successfully using only constraints for the set $\boldsymbol{P}_2(G)$, in which case the constraints have the following form:



$$e_{ij} + e_{jk} \geq d(P)(x_{ij} + x_{jk} - 1)$$

Adding these to the TSP integer program would only make the solution harder to obtain. To obtain lower bounds in a reasonable time the PTMP LP bound is used with the extra constraints.

---

[5] The length of the path is defined to be the number of edges.

[6] This is partly due to the increased solution time needed for the mathematical programming with the additional constraints. However, a large proportion of the run time is taken to formulate all of the FAP constraints as the deficit of every possible path in the constraint graph must be checked.

Minimise: $\sum_{ij \in E(G_0)} c_{ij} x_{ij} + \sum_{ij \in E(G)} e_{ij}$

Subject to:

$\sum_{j:ij \in E(G_0)} x_{ij} = 2$	$i \in V(G_0)$	Degree constraints
$E_P \geq d(P)\left(X_P - \left(\lvert E(P) \rvert - 1\right)\right)$	$P \in \boldsymbol{P}_2(G)$	FAP constraints
$x_{ij} \in \{0,1\}$	$ij \in E(G_0)$	Integrality constraints
$e_{ij} \in \{0,1,2,\ldots\}$	$ij \in E(G_0)$	

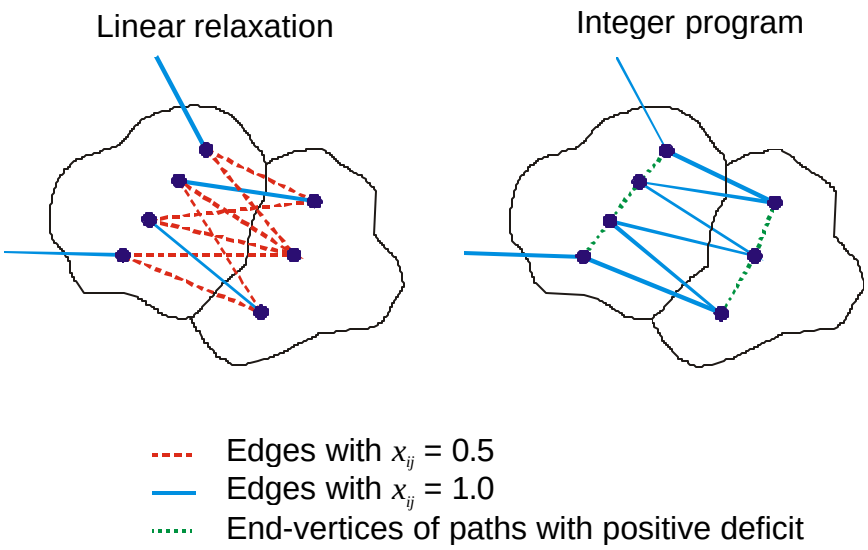
PTMP FAP IP

The results in [16] show that over a range of realistic data sets the PTMP_LP bound is within 1% of the TSP bound, and the PTMP_FAP_LP is typically between 0.2% and 3% better than the TSP bound, depending on the problem type[7]. The time taken to calculate the bounds is also significantly less than that of the TSP bound.

Bounds for cellular problems

For non-cellular problems, the effect on the solution of the relaxation from an integer program to a linear program is generally small. This is true for both the PTMP and PTMP FAP bounds. For cellular problems, the relaxation is good for the PTMP bound. However, the relaxation for the PTMP FAP is particularly bad, and in most cases negates the effect of the additional FAP constraints. This is because all edges between any two cells have equal weights c_{ij} and so typically, the x_{ij} variables take non-integral values that avoid any non-zero excess values being incurred. This is demonstrated in Example 1.

[7] 0.2% for cellular problems and 3% for non-cellular problems.

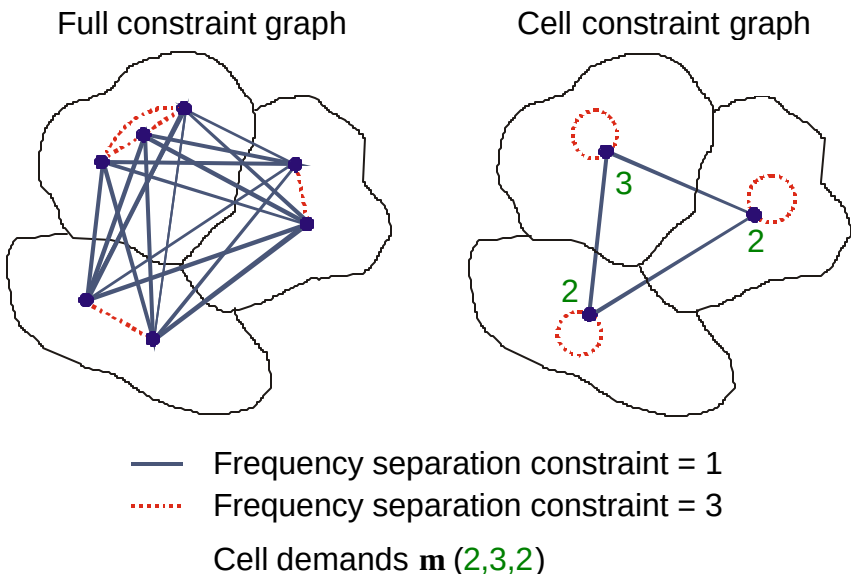


In this example, the constraint between transmitters in the same cell is 3, and between transmitters in different cells is 1. The linear relaxation of the PTMP_FAP bound has a solution with value 7 for internal edges. The integer programming solution has value 9 as there are 3 edge disjoint paths with a deficit of 1, hence at least 3 edges must be assigned an excess value of 1.

Example 1

By utilising the special structure of cellular problems the gap between linear and integer programs can be reduced. The cellular structure gives rise to a *cellular constraint graph* $\overline{G}$ in which the vertices correspond to cells, and the edges correspond to the frequency separation constraint between the two cells. Specifically, each edge ij in the cellular constraint graph has a weight $\overline{c}_{ij}$ that is equal to the edge weight c_{rs} in the full constraint graph, where v_r corresponds to any transmitter in cell i and v_j corresponds to any transmitter in cell j. Whereas the full constraint graph is simple[8], the cellular constraint graph will have loops corresponding to the cosite constraint between transmitters in the same cell. Each vertex v_i of the cellular constraint graph is labelled with the required number of transmitters, or demand, denoted by m_i, of the corresponding cell. Example 2 shows the construction of a cell constraint graph from a full constraint graph.

[8] A simple graph has no loops or multiple edges.



Example 2

The integer program for finding a minimum weight Perfect Two Matching can be simplified by using the cell constraint graph $\overline{G}$. Variables S_{ij} are introduced to represent the number of edges in a perfect two matching in G between vertices corresponding to cells i and j, as shown below. As before, a dummy vertex v_0 is added to the cell constraint graph joined by a single edge of weight 0 to each other vertex to give the graph $\overline{G}_0$. The demand of the dummy vertex is taken to be 1.

Minimise:	$\sum_{ij \in E(\overline{G}_0)} \overline{c}_{ij} S_{ij}$				
Subject to:	$\sum_{j:ij \in E(\overline{G}_0)} S_{ij}$	$=$	$2m_i$	$i \in V(\overline{G}_0)$	Degree constraints
	S_{ij}	$\in$	$\{0,1,\ldots, 2\min(m_i,m_j)\}$	$ij \in E(\overline{G}_0)$	Integrality constraints

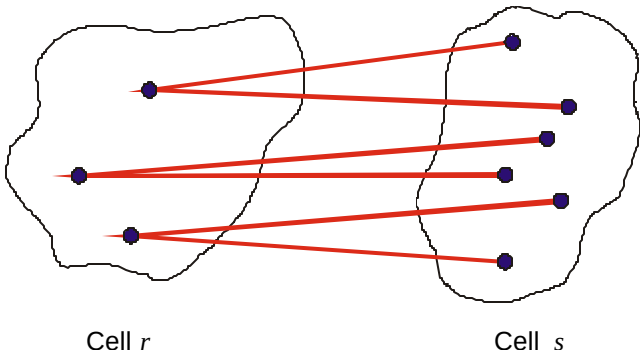
PTMP IP (CELLULAR)

Any solution to this integer program can be used to construct a perfect two matching of minimum weight in the full constraint graph G. This can be done using a simple greedy algorithm.

This cellular representation does not allow equivalent constraints for all paths in P(G) to be added conveniently. However, experimentation suggests that the

most important constraints are based on the cosite constraints, and these constraints can be easily incorporated.

Suppose $S_{rs} > m_r$ in a solution to PTMP IP (CELLULAR). Then any perfect two matching in G_0 constructed from this solution must contain at least $S_{rs} - m_r$ edge-disjoint paths of the form:



If $\overline{c}_{ss} - 2\overline{c}_{rs} > 0$ then each of these paths in G_0 has positive deficit, and so requires an excess value of $\overline{c}_{ss} - 2\overline{c}_{rs}$. Since the paths are edge-disjoint, the total required excess on all paths must be at least $(\overline{c}_{ss} - 2\overline{c}_{rs})(S_{rs} - m_r)$. Hence if E_{rs} is the total excess required on edges between cells r and s, the following constraint must be satisfied:

$$E_{rs} \geq (\overline{c}_{ss} - 2\overline{c}_{rs})(S_{rs} - m_r) .$$

Similarly, by considering S_{rs} - m_s, the following constraint is obtained:

$$E_{rs} \geq (\overline{c}_{rr} - 2\overline{c}_{rs})(S_{rs} - m_s) .$$

Incorporating these constraints for each pair of cells into the PTMP IP (CELLULAR) gives the following integer program.

Minimise: $\sum_{ij \in E(\overline{G}_0)} \overline{c}_{ij} S_{ij} + E_{ij}$

Subject to:	$\sum_{j:ij \in E(\overline{G}_0)} S_{ij} = 2m_i$	$i \in V(\overline{G}_0)$	Degree constraints
	$E_{rs} \geq (\overline{c}_{rr} - 2\overline{c}_{rs})(S_{rs} - m_s)$	for $\overline{c}_{rr} - 2\overline{c}_{rs} > 0$	FAP constraints
	$E_{rs} \geq (\overline{c}_{ss} - 2\overline{c}_{rs})(S_{rs} - m_r)$	for $\overline{c}_{ss} - 2\overline{c}_{rs} > 0$	
	$S_{ij} \in \{0,1,..., 2\min(m_i,m_j)\}$	$ij \in E(\overline{G}_0)$	Integrality constraints
	$E_{ij} \in \{0,1,...\}$	$ij \in E(\overline{G}_0)$	

PTMP FAP IP (CELLULAR)

Section 2 – Subproblems for lower bounds

Choice of subproblem: cliques

For most frequency assignment problems the constraint graphs contain large numbers of edges with weight zero. If the bounds described are applied to such a constraint graph, they will typically give a bound close to zero. However, better bounds can be obtained by considering a critical subset of the problem. Obviously any lower bound for the span of a subproblem is also a lower bound for the span of the full problem. Cliques[9] have been demonstrated [14,15,16,17] to be a good initial choice for determining bounds, particularly when information about the size of the frequency constraints is included.

Definition A *level-p clique*[10] in a constraint graph G is the largest complete subgraph in which each edge has label at least $p+1$, that is contained in no larger such subgraph.

If the bounding techniques are applied to level-p cliques ($p \geq 0$), then there are no edges with label 0, and the resulting bound is generally better than that for the full problem. Normally the best procedure is to choose the largest level-p clique in the constraint graph for each value of p, and find the best bound obtained for each of them.

Definition A *maximal level-p clique* in a constraint graph G is the level-p clique that contains the largest number of vertices.

To find the maximal clique in a graph is an NP-complete problem, however for constraint graphs arising from frequency assignment problems it can often be solved in reasonable time [5]. This is based on using an algorithm of Carraghan and Pardalos [10] together with orderings of the transmitters based on the set of constraints that involving each transmitter. Solutions can usually be obtained by

[9] A clique in a graph is a complete subgraph that is contained in no larger complete subgraph.

[10] Note that this definition is slightly modified from those that appear in [1,16,17] due to the modified definition of constraint graph.

this method for problems with up to 800 transmitters. Although this is too restrictive for most realistic cellular problems, by considering *weighted cliques* in the cellular constraint graph problems with up to 800 cells can be handled (typically this may involve up to 8,000 – 10,000 transmitters).

Definition A *level-p weighted clique* in a cellular constraint graph $\overline{G}$ is the level-p clique such that all loops have labels at least $p+1$ (i.e. all cosite constraints are at least $p+1$), with the largest sum of demands.

Choice of subproblem: Extending cliques based on partial assignments

If the lower bound obtained from a clique matches the value of an assignment of the full problem then a tight bound has been obtained. If not, either the bound or the assignment needs to be improved further. It is possible that the bound could be improved further by considering longer paths in the FAP constraints. However, the effect of paths of length 3 or more is usually negligible when linear relaxations are used. Another possibility is that the gap between the linear and integer programs is causing the difference. Both of these cases would require the solution of an integer program to improve the bound. Alternatively, it could be that the clique does not capture the full difficulty of the problem. Often the bound obtained from a clique is tight for the subproblem but not for the full problem. In this case, an assignment of the clique alone can be made with span matching the bound, but no assignment can be found for the full problem with the same span. If the bound for the clique is not tight, a better subproblem can often be found by successively adding new vertices to the clique. The choice of these additional vertices is critical. A successful method of choosing these additional vertices is the following iterative procedure [16,7] (two further methods are described in [7]):

1. Generate a maximal level-p clique as the current subproblem.
2. Assign the current subproblem.
3. For each vertex outside of the subproblem, consider the number of frequencies that can be assigned to the vertex without introducing constraint violations with vertices within the subproblem.

4. Add the vertex with the smallest number of available frequencies to the current subproblem. For cellular problems all vertices corresponding to the same cell are also added.
5. Evaluate the lower bound on the subproblem and if the bound is still not tight, repeat from step 2.

Results in [16,7] show that this method can sometimes be successful in achieving tighter bounds than those obtained from a clique alone.

Heuristic generation of subproblems

The method of extending cliques described in the previous section may be impractical in general as:

1. The method can be time consuming, as it requires repeated frequency assignments to be generated.
2. It is dependent on the assignments obtained. It may be that the vertex identified by steps 3 and 4 is only critical with respect to the particular assignment found in step 2.
3. It provides no mechanism for removing vertices that are detrimental to the bound.
4. The addition of some vertices may cause the bound to reduce.
5. The method requires an initial clique to be found. A maximal clique is the obvious choice, however there is no guarantee that the largest clique will lead to the best bound.
6. After a certain number of vertices have been added the bound starts to reduce quickly as the number of edges with weight 0 increases.

An alternative method is to use a metaheuristic algorithm that attempts to generate a subproblem that gives the best lower bound for the problem. A brief introduction to metaheuristic algorithms is given here, full details of the particular algorithm used to generate the results presented (simulated annealing) are given in Section 4.

Many metaheuristics work on the principle of local search. An initial configuration is chosen and successive configurations are selected from a set of

those similar to the current one. For example, for this problem, the neighbourhood of a subproblem may consist of all subproblems that can be obtained by adding or removing a single cell/transmitter. At each iteration a new configuration is selected from the neighbourhood of the current configuration, and tested against some criteria, for example, the value of a cost function. If the configuration is accepted the search continues from the new configuration, otherwise the new configuration is rejected and the search continues from the previous configuration. This process continues until some termination criteria are met, for example, when a fixed number of configurations have been tested, or if no improvement has been made for a given number of iterations.

For the heuristic generation of subproblems for the lower bounds, two types of neighbourhood have been considered. A site refers to a single cell in a cellular problem and to a single transmitter in non-cellular problems.

Random move. A site is chosen at random. If it is already in the subproblem it is removed, otherwise it is added to the current subproblem.

Connected move. A site is chosen at random from those sites that are either contained in the current subproblem or constrained with some site in the current subproblem. If it is already in the subproblem it is removed, otherwise it is added to the current subproblem.

The cost function used to evaluate each subproblem consists of a calculation of one of the bounding methods:

- PTMP LP
- PTMP FAP LP
- PTMP IP
- PTMP FAP IP

In practise the integer programming bounds are far too computationally expensive to be used.

The algorithms described can also be used with a cost function involving the size of the subproblem being considered, with the aim of encouraging the search to remove cells that have no effect on the bound. However, the effects on the bounds produced using this method appear to be insignificant. This type

of cost function may be of use in finding large subproblems with large bounds. These can potentially be used as part of the assignment procedure.

To use metaheuristic algorithms effectively it is essential to have a cost function that can be quickly and easily calculated as this will limit the total number of configurations that can be evaluated. The PTMP LP bound described earlier satisfies this property. For the results presented in this report solutions to the linear programs were obtained using the commercial mathematical programming package CPLEX[11]. The times taken in seconds to calculate the bounds for typical subproblems are:

	PTMP LP	PTMP FAP LP
32 cell clique in cellular problem	0.11	0.12
45 transmitter clique in non-cellular problem	0.23	1.41
45 randomly chosen transmitters in non-cellular problem	0.14	3.81

The non-cellular PTMP FAP LP bounds take considerably longer than equivalent sized cellular problems. This is due to the larger number of FAP constraints involved, for an N cell problem there are at most N*(N-1) FAP constraints, but for an N transmitter non-cellular problem there are at most N*(N-1)*(N-2)/2. This is also reflected in the time taken to formulate the constraints (the times shown above are the times for the solution to be obtained after it has been formulated). In both cases each possible constraint must be tested to see if the path has a positive deficit.

Using the PTMP LP bound as a cost function in a metaheuristic to generate subproblems gives Algorithm 1.

[11] http://www.cplex.com/

```
bestBound    = 0                              // Reset overall bound and run counter.
currentRun   = 0

WHILE ( currentRun < totalRuns )

    currentRun++                              // Increment run counter.
    subproblem.initialize                     // Load initial configuration.

    WHILE NOT ( metaheuristic.isFinished )    // Until termination criteria are met.

        subproblem.perturb                    // Apply move and evaluate
        cost = subproblem.ptmp_lp             // the cost function for the
        metaheuristic.step( cost )            // new configuration using the
                                              // chosen metaheuristic.
        IF ( metaheuristic.isAccept )

            IF cost > bestBound               // Update best overall bound if
                bestBound = cost              // necessary.
            END IF

        ELSE

            subproblem.reset                  // Undo the last move.

        END IF

    END WHILE

END WHILE
```

Algorithm 1

For cellular problems the PTMP_FAP_LP bound can be calculated quickly enough to allow it to be used in place of the PTMP LP bound to test every configuration.

For non-cellular problems the PTMP_FAP_LP bound is more time consuming and so for larger problems it is impractical to use as the cost function to test every configuration. Instead, the PTMP_LP bound is used to guide the search. Additionally, whenever the PTMP LP bound for a configuration is within a specified percentage of the best PTMP LP bound found so far, the PTMP_FAP_LP bound can be applied to the subproblem[12]. This method is shown in the Algorithm 2. Note that this algorithm will also give the best bound obtained by Algorithm 1 (as bestPtmpBound).

[12] All of the experimental results suggest a strong correlation between the PTMP LP and PTMP FAP LP bounds, so it is reasonable to use the PTMP LP to guide the search for the best PTMP FAP bound. Ideally, a threshold value that is strictly less than 100% should be used as this will ensure every configuration with the maximum PTMP bound will be tested by the PTMP FAP bound.

```
bestPtmpBound = 0                              // Reset best PTMP_LP bound (used to
bestBound     = 0                              // guide the search), best overall
currentRun    = 0                              // bound and run counter.

WHILE ( currentRun < totalRuns )

    currentRun++                               // Increment run counter.
    subproblem.initialize                      // Load initial configuration.

    WHILE ( NOT metaheuristic.isFinished )     // Until termination criteria are met.

        subproblem.perturb                     // Apply move and evaluate
        cost = subproblem.ptmp_lp              // the cost function for the
        metaheuristic.step( cost )             // new configuration using the
                                               // chosen metaheuristic.
        IF ( metaheuristic.isAccept )

            IF ( cost > bestPtmpBound )        // Update best PTMP_LP bound if
                bestPtmpBound = cost           // necessary.
            END IF

            IF ( subproblem.ptmp_lp > threshold*bestPtmpBound )
                                               // If PTMP_LP bound is within specified
                                               // percentage of the best so far,
                                               // calculate the PTMP_FAP_LP bound for
                                               // the new configuration and update as
                                               // necessary
                cost = subproblem.ptmp_fap_lp
                IF ( cost > bestBound )
                    bestBound = cost
                END IF

            END IF

        ELSE

            subproblem.reset                   // Undo the last move

    END WHILE

END WHILE
```

Algorithm 2

Metaheuristics generally shows a rapid improvement in the cost function initially, while improvement later takes longer. It is therefore beneficial to limit the time taken by the metaheuristic on a single run and instead perform many runs, selecting the best bound overall**.** It is possible to use an initial configuration to start the search with a good subproblem, for example a clique. In practise, this requires a more careful tuning of the parameters for the chosen metaheuristic. They must be set so that enough solutions are accepted initially to allow the search to move away from the local maximum; but the setting must also ensure that enough solutions are rejected so that the subproblems considered do not grow too large. Bounds for large subproblems take longer to compute and generally have a relatively large number of edges of weight zero, which leads to lower bounds. Better results are obtained by selecting no cells

initially, as in this case the parameters can be set conservatively in order to prevent the subproblems growing too large. Although many of the runs will give a bound that is worse than that of the maximal clique, by selecting an appropriate number of runs a better bound will often be obtained as more of the search space will be explored. This is illustrated in Figure 1, which shows the results of 50 runs of Algorithm 1 on a typical problem.

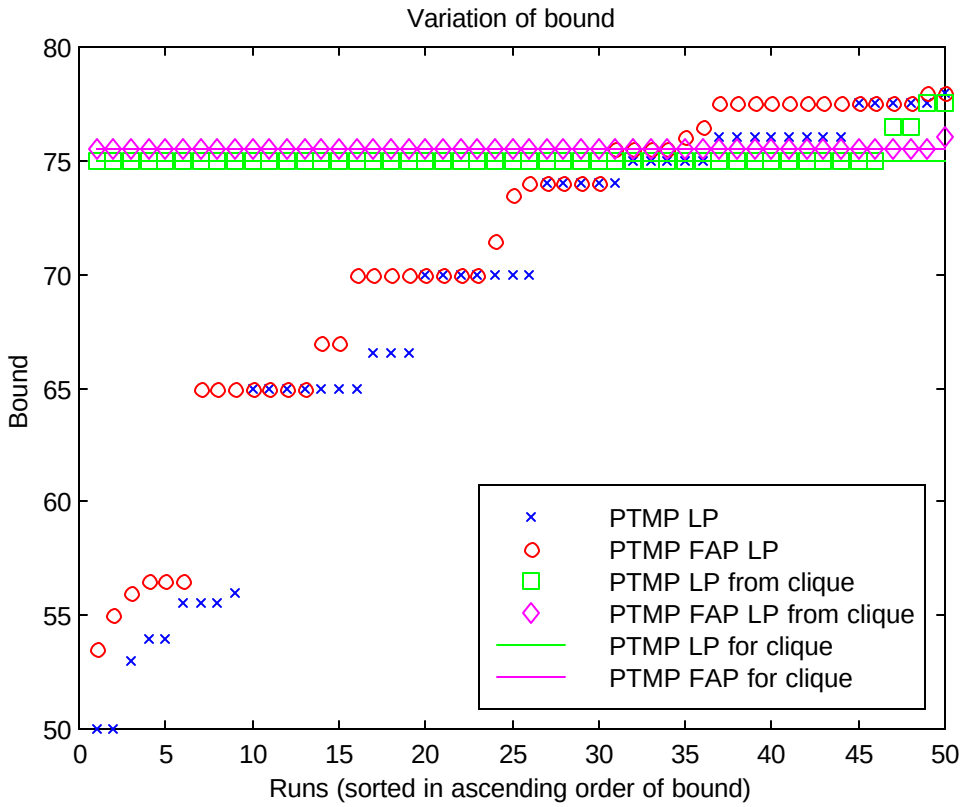


Figure 1

Connected moves were found to give the best bounds in a given amount of time, particularly when starting with no initial subproblem[13]. They also require less tuning of parameters to get the best results. Moves consisting of adding/removing multiple sites at a time were also considered but had no positive effect and for some problems had a negative effect. All results presented in this report use a connected move, changing only one site in each move, unless stated otherwise.

[13] See page 32.

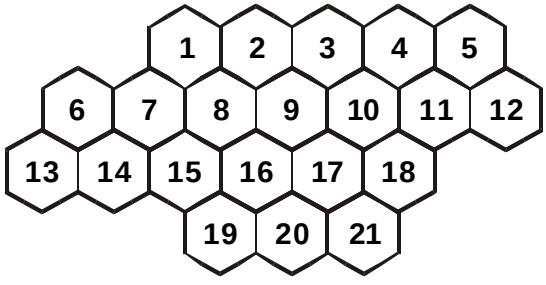
Section 3 – Results

The results presented in this section demonstrate the use of Algorithms 1 and 2. The metaheuristic used is simulated annealing. Full details of simulated annealing and the implementation used can be found in Section 4. The table headings used to present the results in this section are as follows:

Number of transmitters in maximal level-0 clique	The size of the largest level-0 clique. Calculated using an algorithm of C&P [10] using orderings of the transmitters or cells [5].
Extended clique bound	A bound obtained by applying the TSP bound to a clique or a clique that has been extended by the successive addition of vertices. Where noted, the PTMP FAP LP bound has been used instead of the TSP bound.
Algorithm 1 (PTMP LP)	The best bound obtained using Algorithm 1 over 50 individual runs.
Algorithm 2 (PTMP_FAP LP)	The best bound obtained using Algorithm 2 over 50 individual runs.
Upper bound	The span of the best known assignment. All assignments are made using FASoft [6], a package for frequency assignment based on several metaheuristic algorithms.

Philadelphia data sets

A well-studied range of benchmark problems is the so-called *Philadelphia* problems. These are based on a theoretical cellular network around Philadelphia and have appeared extensively in the literature [9,7]. The cellular structure is shown below.



The demand (number of transmitters) in each cell is given by one of the following requirement vectors

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
D1	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
D2	5	5	5	8	12	25	30	25	30	40	40	45	20	30	25	15	15	30	20	20	25
D3	8	25	8	8	8	15	18	52	77	28	13	15	31	15	36	57	28	8	10	13	8

The constraints between each pair of transmitters are characterised by six distances d_0 ,…,d_5. If d is the distance between transmitters i and j then

$$\left| f(v_i) - f(v_j) \right| \geq 0 \qquad \text{if } d_0 \pounds d,$$

$$\left| f(v_i) - f(v_j) \right| \geq k \qquad \text{if } d_k \pounds d < d_{k-1}, \text{ where } k = 1,...,5.$$

where all of the transmitters are placed at cell centres and the distance between adjacent cell centres is 1.

The following constraint sets have appeared in the literature.

	d_0	d_1	d_2	d_3	d_4	d_5
C1 [7]	$\sqrt{7}$	$\sqrt{3}$	1	1	1	0
C2 [7]	$\sqrt{12}$	$\sqrt{3}$	1	1	1	0
C3 [9]	$\sqrt{12}$	2	1	1	1	0

The use of Algorithms 1 and 2 for all combinations of requirement vector and constraint set is shown below.

Requirement vector	Constraint set	Number of transmitters in maximal level-0 clique	Extended clique bound	Algorithm 1 (PTMP LP)	Algorithm 2 (PTMP_FAP LP)	Upper bound
D1	**C1**	140	178	177	177	179
D1	**C2**	240	239	239	239	239
D1	**C3**	240	239	239	239	259
D2	**C1**	180	252	252	252	252
D2	**C2**	258	257	257	257	257
D2	**C3**	258	258	290	300	324
D3	**C1**	275	426	426	426	426
D3	**C2**	360	426	426	426	426
D3	**C3**	360	524[14]	483	524	524

Note that for all problems, the best bounds are matched by Algorithm 2, apart from D1 C1. Here the best bound of 178 is obtained by a theoretical argument specific to this problem.

[14] Using PTMP FAP LP.

Other cellular data sets

The results below are for a series of real cellular problems with up to 10,000 transmitters.

Data set	Number of transmitters in maximal level-0 clique	Extended clique bound	Algorithm 1 (PTMP LP)	Algorithm 2 (PTMP_FAP LP)	Upper bound
One	285	284	284	284	306
Two	282	336	336	336	336
Three	363	362	373	383	411

The following data sets were generated from random transmitter site placements made using the technique described in [17] using constraint generation techniques described in [12][15]. A single transmitter is placed at each site initially, giving the following results.

Data set	Number of transmitters in maximal level-0 clique	Extended clique bound	Algorithm 1 (PTMP LP)	Algorithm 2 (PTMP_FAP LP)	Upper bound
Random1a	12	11	11	**11**	**11**
Random1b	26	25	25	**25**	**25**
Random2a	8	7	7	**7**	**7**
Random2b	14	13	13	**13**	**13**
Random3a	10	9	9	**9**	**9**
Random3b	12	11	11	**11**	**11**
Random4a	9	8	8	**8**	**8**
Random4b	15	14	14	**14**	**14**
Random5a	11	10	10	**10**	**10**
Random5b	18	17	17	**17**	**18**

[15] The data sets 1,2,3,4,5 correspond to networks 1,5,6,7,8 respectively. Data sets 'a' are constructed using a signal to interference ratio of 9dB and 'b' are calculated using 17dB.

The data sets were also evaluated with a set of random demands, generated by assigning a random integer from the range [1,10] to each site.

Data set	Number of transmitters in maximal level-0 clique	Extended clique bound	Algorithm 1 (PTMP LP)	Algorithm 2 (PTMP_FAP LP)	Upper bound
Random1c	66	65	65	**65**	**65**
Random1d	150	149	148	**148**	**149**
Random2c	38	37	37	**37**	**37**
Random2d	76	75	75	**75**	**75**
Random3c	56	55	55	**55**	**55**
Random3d	63	84	84	**84**	**86**
Random4c	44	43	43	**43**	**43**
Random4d	70	69	69	**69**	**70**
Random5c	66	65	65	**65**	**65**
Random5d	115	114	114	**114**	**122**

Fixed radio links data sets

The data in this section is constructed to represent realistic fixed radio links problems, based on a military radio structure.

Data set	Number of transmitters in maximal level-0 clique	Extended clique bound	Algorithm 1 (PTMP LP)	Algorithm 2 (PTMP_FAP LP)	Upper bound
T95a	23	47	47	**47**	**47**
T95b	23	47	47	**47**	**48**
T252a	44	50	53	**53**	**58**
T252b	44	50	53	**53**	**60**
T282a	45	75	78	**78**	**99**
T282b	45	75	80	**81**	**102**
T410a	64	114	116	**117**	**143**
T410b	64	114	116	**118**	**145**
T450a	88	94	90	**92**	**122**
T450b	88	94	90	**92**	**116**
T490a	79	126	132	**133**	**187**
T490b	79	126	132	**133**	**185**
T726a	99	171	182	**182**	**226**
T726b	99	181[16]	182	**184**	**247**

[16] This is after extensive extension of a clique using the method described in [16]. The data set is used as example links8 in [16].

The extended clique bounds for these problems are derived from level-0 and level-1 cliques.

Section 4 – Implementation

The Algorithms 1 and 2 (as described in Section 2) have been implemented in C++ using an objected oriented simulated annealing implementation as the controlling metaheuristic. The simulated annealing code was developed jointly with S.J. Chapman.

Simulated annealing: General description

The simulated annealing metaheuristic is based on the physical process of annealing. A temperature is used to control the ability to move away from local maxima that are not global maxima[17]. The temperature is given a high value initially that reduces during the course of the algorithm. At each iteration, the current configuration with cost C_{new} is tested against the last accepted configuration with cost C_{accept}. The current configuration is accepted if either

- $C_{new} > C_{accept}$, (that is, if the current configuration is an improvement)
- or $e^{\frac{C_{new} - C_{accept}}{kT_{SA}}} > random$, where T_{SA} denotes the current temperature and ***random*** is a random number from [0,1). The constant k is used to scale the probability of accepting a worse solution and should be set based on the problem data. That is, a worse configuration is accepted with a probability dependent on the temperature. As the temperature reduces, so does this probability. The probability is also dependent on the quality of the new configuration; the lower the cost of the new configuration, the less likely it is to be accepted. Accepting a worse configuration is referred to as a *tunnel* event.

After N_{trial} configurations have been tested at a temperature T_{SA} it is reduced to a new temperature $\phi(T_{SA})$. The value of N_{trial} and the function $\phi(T_{SA})$ depend on the cooling scheme used (see Table 1).

Logarithmic cooling differs from the other cooling schemes in that it sets it's own starting temperature. For the others an initial value must be set.

[17] This description of simulated annealing is for a maximization problem.

The algorithm terminates when either

1. The temperature reaches a user specified minimum.
2. A given number of temperatures have passed with no improvement. These are referred to as *frozen* temperatures.
3. A fixed number of iterations have been tested (logarithmic cooling only).

Cooling scheme	**$f(T_{SA})$**	**N_{trial}**	**Constants**
Linear	$T_{SA} - \alpha_1$	constant	$\alpha_1 \geq 0$
Geometric	$\alpha_2 T_{SA}$	constant	$\alpha_2 \in [0,1]$
Costa	$\alpha_3 T_{SA}$	$N_{trial}^{new}(T_{SA}) = \min\left(\frac{N_{trial}(T_{SA})}{a_3}, N_{\max}\right)$	$\alpha_3 \in [0,1]$ N_{max} (upper limit on iterations at each temperature)
Lundy and Mees	$\frac{T_{SA}}{1 + a_4 T_{SA}}$	1	α_4 small valued
Hurley and Smith	$T_{SA}\left(1 + \frac{\ln(1+a_5)T_{SA}}{3s(T_{SA})}\right)^{-1}$ $s(T_{SA}) = \frac{1}{N_{trial}}\sqrt{\Phi(T_{SA})}$ $\Phi(T_{SA}) = N_{trial}\sum_{i=1}^{N_{trial}}[C^i(T_{SA})]^2 - \left(\sum_{i=1}^{N_{trial}} C^i(T_{SA})\right)^2 + 0.5$ $C^i(T_{SA})$ is the cost of the i^{th} trial network at temperature T_{SA}	constant	α_5
Logarithmic	$width^{\left(\frac{N_{drop}}{cold} - i\right)}$ where $i=1$ to $N_{drop.}$	$N_{trial} = N_{\log}^{(1-g)}$ $N_{drop} = N_{\log}^{g}$	$g \in [0,1]$ N_{log} total iterations width, cold used to set temperatures

Table 1

Parameter tuning

All the results presented in this report were generated using the logarithmic cooling schedule with the parameters:

N_{log}	3000
γ	0.1
width	2
cold	1

Logarithmic cooling is used to avoid generating initial temperatures. The number of runs should be at least 25, preferably between 50 and 100.

The only parameter that remains to be set is k, which effectively controls the amount of tunnelling that occurs (for fixed values of cold and width). Figures 2--5 demonstrate how variation in k can affect the bounds obtained. Each figure shows a single run of Algorithm 1 for a non-cellular problem with 282 transmitters. Apart from the value of k, all parameters are as above. Each figure contains two graphs showing:

1) The variation in the PTMP LP bound for each accepted subproblem, against the iteration at which it was accepted.
2) The variation in the size of each accepted subproblem, against the iteration at which it was accepted.

Note that in each case the choice of k is critical. For higher values of k ($k \geq 0$) the bound quickly reduces to 0, and the subproblem size grows to around 140 transmitters.

Once either Algorithm 1 or 2 accept a subproblem with bound 0, it is very hard for the simulated annealing to guide the search back to a positive bound. This is because a move to a subproblem with equal cost is always accepted as a tunnel event[18]. The random nature of the move means that vertices are likely to be added, and accepted, that do not increase the bound. This continues until each subproblem consists of approximately half of the vertices. At this point the

[18] Since $e^{\frac{0}{kT_{SA}}} = 1$.

probability of a random move adding a site is approximately equal to the probability of a site being removed from the subproblem. Thus the size of the subproblem remains relatively constant. Using a connected move instead of a random move improves the situation, making the choice of k less critical. Another potential improvement is to disallow tunnel events when the bound is 0 (although tunnelling must be allowed to occur for small subproblems to avoid getting stuck at a subproblem consisting of a single vertex). However, in practise this has little effect and such a method would need more refinement.

Figures 6--9 show similar results for a cellular problem with 742 cells (8166 transmitters). Again the choice of k can be critical. These figures also demonstrate the disadvantage of starting from an initial configuration. The bounds at the end of the run are no better than those at the start. If the cells within the subproblems are examined, they show considerable overlap (almost all of the cells in the final subproblem are contained in the initial subproblem). This is despite a large number of tunnel events. Thus in order to obtain subproblems that are significantly different, even more tunnel events must be allowed. It is more effective to start with no initial subproblem in each run.

The results in Section 3 are all generated using a value of 0.001 for k. Although this may not be the best value of k for some problems, it has the advantage of being widely applicable in a robust manner.

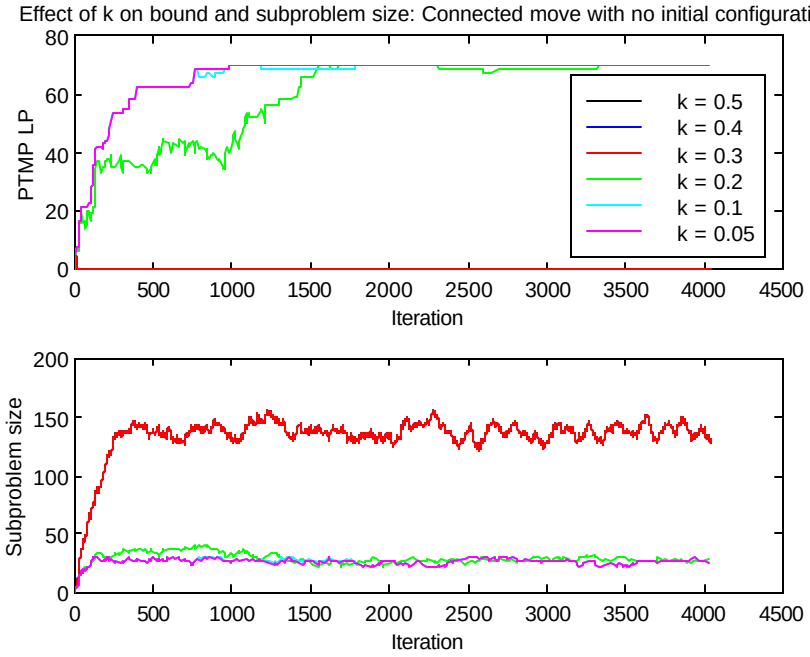


Figure 2

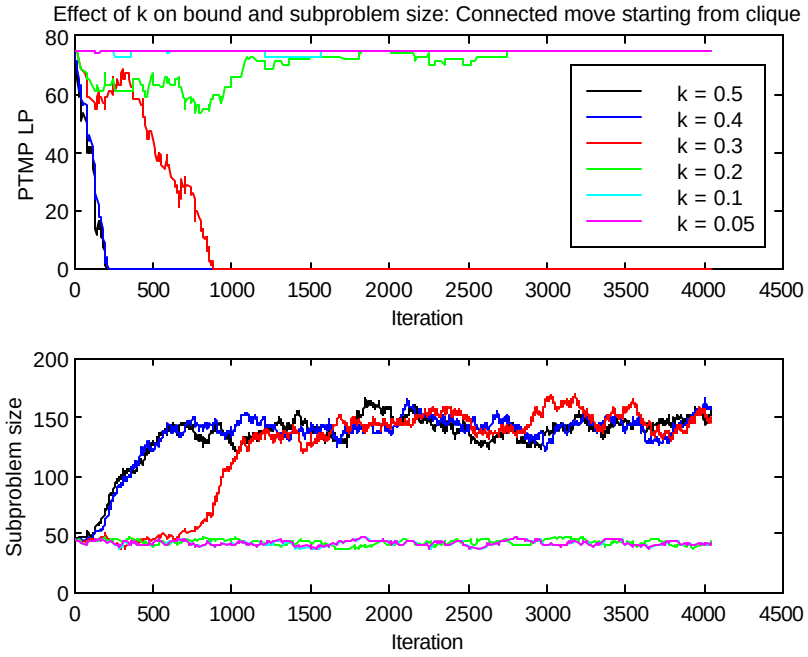


Figure 3

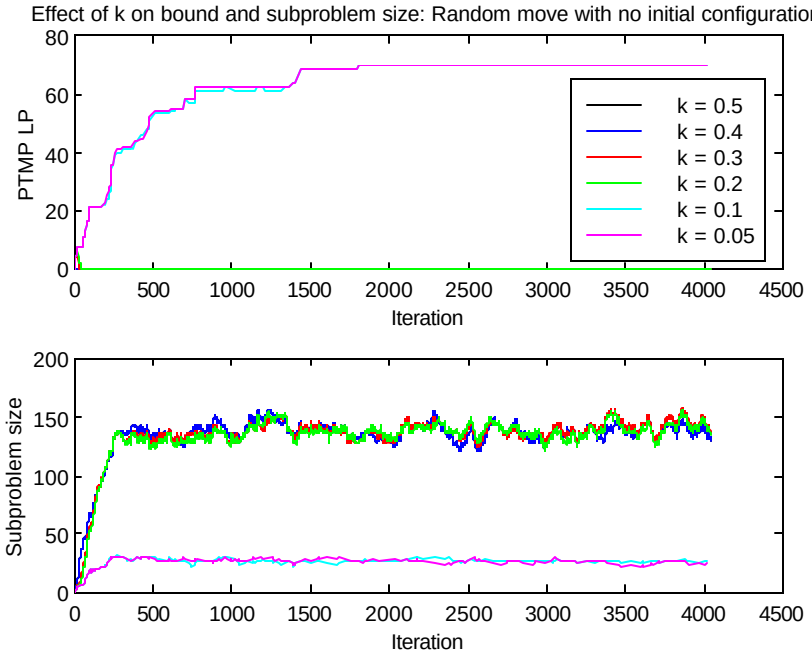


Figure 4

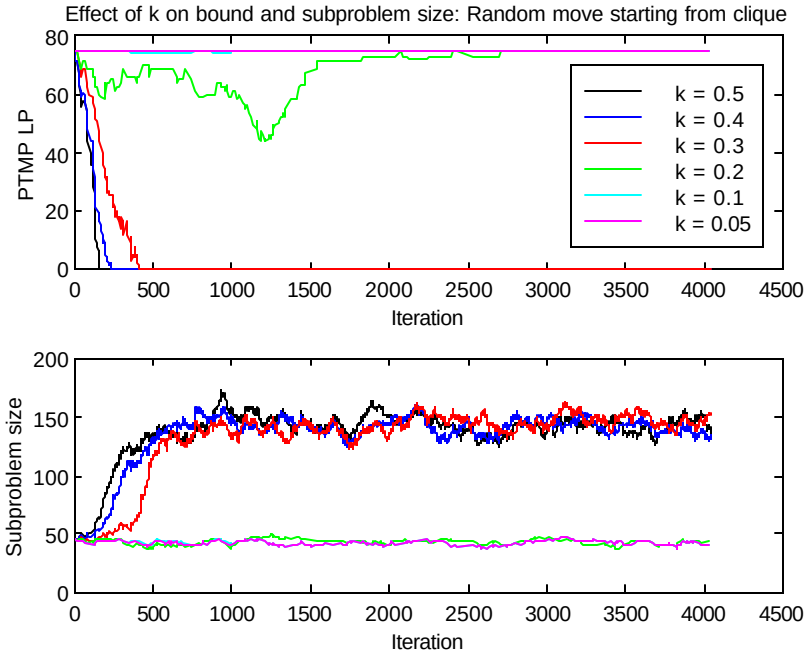


Figure 5

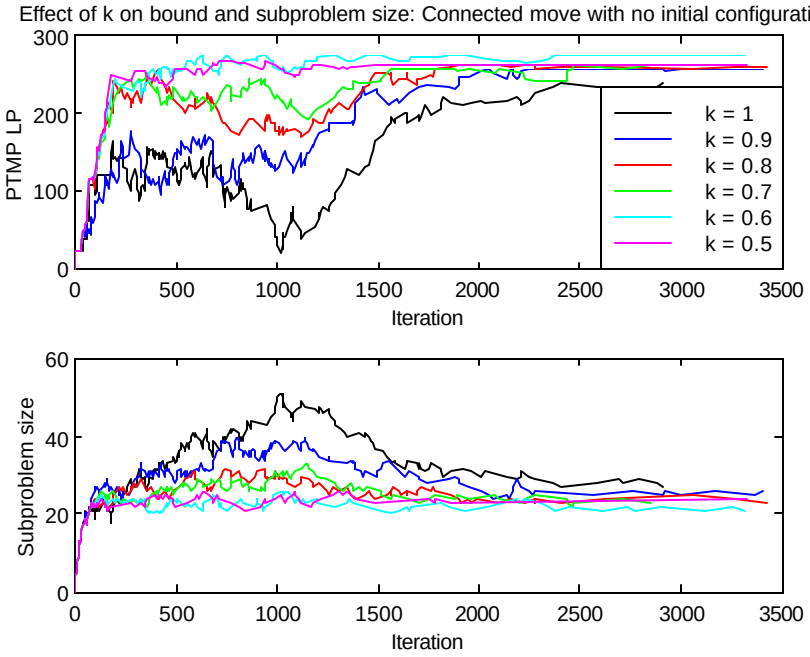


Figure 6

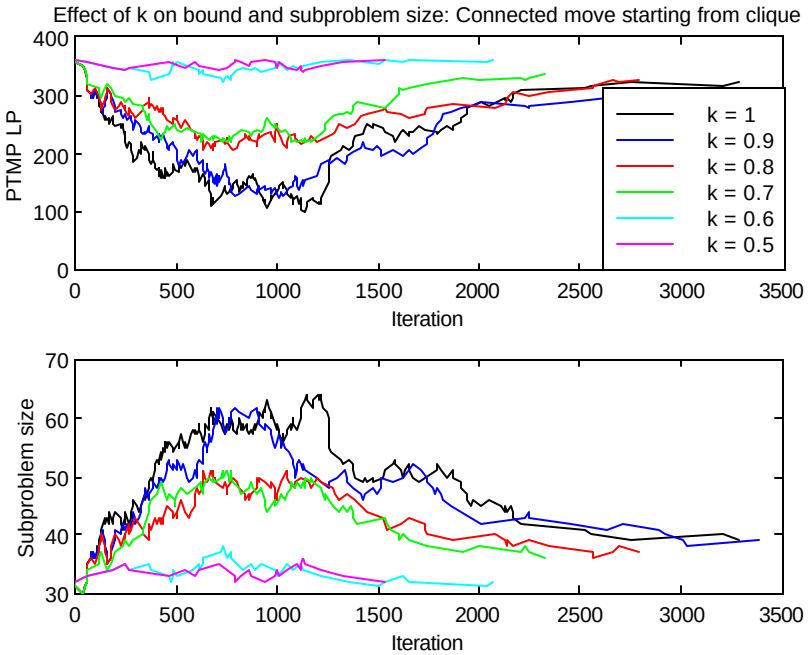


Figure 7

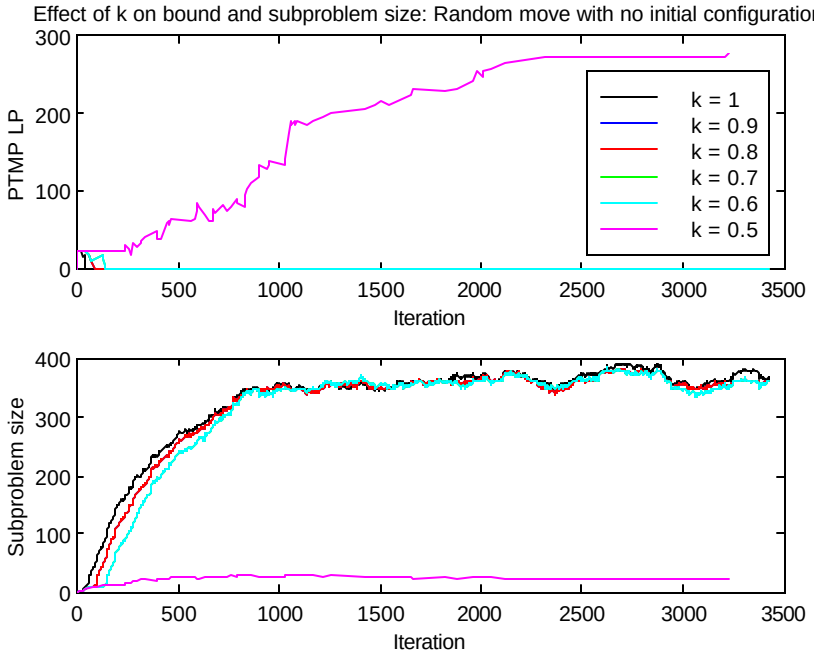


Figure 8

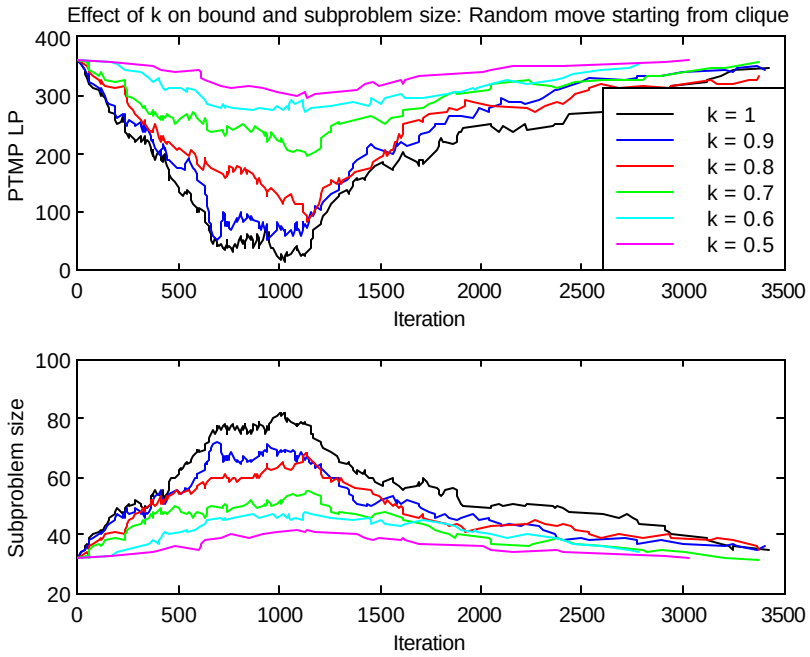


Figure 9

Conclusions and extensions

Two algorithms have been presented that give an automatic, simple method for generating lower bounds for minimum span frequency assignment problems. The advantages of these heuristic generation algorithms compared to clique based methods are:

1. **Quality**. In all cases the bounds obtained by Algorithm 2 match or improve those obtained by applying a bound to a clique. The results for Algorithm 2 are also better than the bounds obtained using an extended clique derived by the methods described in [16].
2. **Ease of use**. Previously a clique detection routine was needed to derive the bounds. Although there are good algorithms for generating maximal cliques for constraint graphs [5], to get the best bounds several cliques have to be found. For some problems a level-0 clique will give the best bound, whereas for others a level-1 or above clique is necessary. In general, there is no way of determining in advance which level is required. Algorithm 2 gives good results without the need for clique detection or clique extension.
3. **Robustness**. Algorithms 1 and 2 give good results applied to all problems without modification.

An implementation of Algorithms 1 and 2 has been extensively tested on a wide range of realistic problems. The main points arising from this experimentation are:

1. Connected moves give better results faster than random moves.
2. A single cell/transmitter should be changed in each move. Changing more than one cell needs careful tuning of the metaheuristic parameters and provides no noticeable improvement.
3. Algorithm 2 gives better results (where possible) than Algorithm 1, without a significant increase in run time. A threshold of approximately 95% should be used.
4. No initial subproblem should be used, as this restricts the search too heavily.

5. For the simulated annealing implementation described, the logarithmic cooling schedule gives good results while requiring less tuning of parameters. The parameters shown below were found to be successful for most problems. The value recommended for k is chosen to be robust, giving good results for all problems. For some problems better values of k may be found that encourage more tunnel events (as seen in Section 4 this value can be from 0.005 to 0.5).

N_{log} (total iterations)	γ	width	cold	k
3000	0.1	2	1	0.001

Extensions to the method could involve:

1. Investigation of the importance of integrality constraints. In some cases it can be seen that the bound obtained from PTMP FAP LP is some way below that of PTMP FAP IP. It is impractical to obtain a solution to full integer programs of PTMP FAP IP, however, better bounds may be possible by adding a selection of the integrality constraints. This may be either by identifying the critical integrality constraints or by using some random selection. Experimentation suggests that integrality constraints are the best way to improve these bounds further; extra frequency assignment constraints have little effect.
2. The choice of parameters used for the metaheuristic may be linked to the problem to be solved. For example, the maximum number of iterations may be dependent on the problem size. Presently the parameters have to be set conservatively in order to ensure the best bound is obtained, however it may be possible to improve the run time if a better choice can be made.
3. The use of different metaheuristics to guide the search. In particular *Tabu search* may be useful as it has the facility to guide the search into new areas by considering the cells that have not been included often.
4. The same basic method could be used to try and identify areas of the problem that are hard to assign, for example by using the span of a *sequential assignment* as the cost function.

References

1. Smith, D.H., Allen, S.M., and Hurley, S. Lower Bounds for Channel Assignment, *in Methods and Algorithms for Channel Assignment, edited R. Leese, Oxford University Press*, to appear.
2. Hurley, S. and Smith, D.H. Meta-heuristics and Channel Assignment, *in Methods and Algorithms for Channel Assignment, edited R. Leese, Oxford University Press*, to appear.
3. Allen, S.M., Smith, D.H. and Hurley, S. Lower Bounding Techniques for Frequency Assignment, Discrete Mathematics, to appear.
4. Allen, S.M., Hurley, S., Smith, D.H. and Thiel, S.U. Using Lower Bounds in Minimum Span Frequency Assignment, *Meta-heuristics Advances and Trends in Local Search Paradigms for Optimization, (ed. S. Voss, S. Martello, I.H. Osman and C. Roucairol), Kluwer Academic,* 191—204 (1999).
5. Smith, D.H., Hurley, S. and S.M. Allen. A New Lower Bound for the Channel Assignment Problem, (submitted).
6. Hurley, S., Smith, D.H. and Thiel, S.U. FASoft: A System for Discrete Channel Frequency Assignment, *Radio Science*, **32** No. 5, 1921—1939, (1997).
7. Smith, D.H., Hurley, S. and Thiel, S.U. Improving Heuristics for the Frequency Assignment Problem, *European Journal of Operations Research*, **107**/1, 76—86, (1998).
8. Smith, D.H. and Hurley, S. Bounds for the Frequency Assignment Problem, *Discrete Mathematics*, **167/168**, 571—582 (1997).
9. Tcha, D-w., Chung, Y-j. and Choi, T-j. A New Lower Bound for the Frequency Assignment Problem. *IEEE/ACM Transactions on Networking*, **5**, 34—39 (1997).
10. Carraghan, R. and Pardalos, P.M. An Exact Algorithm for the Maximum Clique Problem. *Operations Research Letters*, **9**, 375—382, (1990).

11. Pekny, J.F. and Miller, D.L. A Staged Primal-dual Algorithm for Finding a Minimum Cost Perfect Two-matching in an Undirected Graph. *ORSA Journal on Computing*, **6(1)**, 68—81, (1994).

12. Watkins, W.J., Hurley, S. and Smith, D.H., Evaluation of Models for Area Coverage, t*echnical report, Department of Computer Science, University of Wales, Cardiff,* (1998).

13. Raychaudhuri, A. Intersection Assignments, T-colourings and Powers of Graphs. *Ph.D. thesis, Rutgers University.*

14. Smith, D.H., Hurley, S. and Thiel, S.U. Frequency Assignment Algorithms. *Technical report, Final Report Year 1, Radiocommunications Agency Agreement ref. RCCM 070.* http://www.cs.cf.ac.uk/User/Steve.Hurley/Ra/year1/year1.html.

15. Smith, D.H., Hurley, S. and Thiel, S.U. Frequency Assignment Algorithms. *Technical report, Final Report Year 2, Radiocommunications Agency Agreement ref. RCCM 070.* http://www.cs.cf.ac.uk/User/Steve.Hurley/Ra/year2/year2.html.

16. Dunkin, N, and Allen, S.M. Frequency Assignment Problems: Representations and Solutions, *technical report M-97-1, Division of Mathematics and Computing, University of Glamorgan,* (1997).

17. Dunkin, N, Allen, S.M., Smith, D.H. and Hurley, S. Frequency Assignment Problems: Benchmarks and Lower Bounds, *technical report UG-M-98-1, Division of Mathematics and Computing, University of Glamorgan,* (1998).

Appendix A: Software manual

Two pieces of software have been developed for obtaining bounds for minimum span frequency assignment problems:

hs	Uses Algorithms 1 and 2 to try and identify the best possible lower bound for a problem
ptmp	Calculates a bound for a single problem or subproblem

Both pieces of software require the commercial linear program solver Cplex. The `path` environment variable must be set to include the location of the file `cplex.dll`.

File formats

```
  1  2 > 0
  1 10 > 1
  1 14 > 0
  1 55 > 3
  2 10 > 0
  2 54 > 1
 10 11 > 0
 10 14 > 0
```

Ctr file format (input)

Each line of the file specifies a frequency separation constraint between a pair of transmitters or a pair of cells. Each transmitter name must occur in the corresponding var file. Tabs are not permitted.

```
  1 0
  2 0
  3 0
 10 0
 11 0
 14 0
 54 0
 55 0
```

Var file format (input)

Each line specifies a single cell/transmitter. The zero has no meaning but ensures compatibility. The cells/transmitters must be in ascending order. Totally blank lines are ignored. Tabs are not allowed.

```
  1 12
  2  9
  3  3  5
 10 10  3
 11 14  5
 14  9
 54 11
 55  6
```

Demand file format (input)

Each line specifies the demand and cosite for an individual cell in the form:

```
cell demand [cosite constraint]
```

The cosite constraint is optional; if none is specified the cosite constraint for the cell is taken from the `DefaultCosite` parameter given in the parameter file. Any cell not specified in the demand file is assumed to have a demand of one. Demand values for cells not contained in the corresponding var file are ignored (so that a demand file for a full problem can be used as the input for subproblems).

```
  6 cells in subproblem
  2
  3
 10
 11
 14
 54
```

Log file format (input and output)

The first line specifies the number of cells or transmitters in the file. Any text after the number is ignored. Each subsequent line gives the name (as an integer) of a cell or transmitter. If the file is used as input, each name must appear on a line in the corresponding var file for the full problem. If the file is generated as an output, each name corresponds to a line in the input var file.

```
% Run 0
2 3 11 54 14 % 23
3 11 54 14 % 24
3 11 54 14 53 % 24.5
3 11 54 14 53 12 % 30
% Bound 30
```

Subproblem file format (output)

Each line specifies a subproblem that has been accepted by the SA (either because of a tunnel event or because of a success event). The bound for the subproblem is given after the '%'. Each run is separated by '% Run x' before, and '% Bound ???' after.

```
Cellular = "True"
CtrFile = "Example.ctr"   // Example comment
VarFile = "Example.var"
DemFile = "Example.dem"
DefaultCosite = 5
K = 0.001
```

Parameter file format (input and output)

All of the parameters for algorithms 1 and 2, together with input and output filenames are given in a parameter file. Each line specifies a single parameter and has the form

```
ParameterName = "ParameterStringValue"
```

for a string valued parameter, and

```
ParameterName = ParameterValue
```

for parameters that have integer or float values. Tabs are not permitted. C++ style comments ('//') can be used. Parameter values given below are case sensitive.

hs – Heuristic Subgraph generator

Command line options

```
hs –p parameter.par [-v] [-o outputfile] [-s subproblemfile]
```

`-p parameter.par`	Specifies the parameter file to be used.
`-v`	Optional. Verbose mode; the result of every configuration evaluated is output. If not present, results are only output when a new subproblem is accepted (either due to a tunnel or success event).
`-o outputfile`	Optional. If present, the output is sent to the specified file as well as the screen. The parameters for each run can also be output to this file by setting the parameter `OutputParameters`.
`-s subproblemfile`	Optional. If present, every accepted subproblem is written to the specified file.

Parameter settings

The settings to be specified in the parameter file are shown in the following tables, in the form:

Parameter name	Description	Type	Range of allowed values

Red parameters are compulsory in all cases.

Green parameters force additional parameters to be set.

`MAX_INT` and `MAX_FLOAT` are as specified in the C++ include files `limits.h` and `float.h`. When compiled using Microsoft Visual C++ on an Intel platform they take the values 2147483647 and 3.402823466e+38F respectively.

Problem input

Cellular	Specifies whether the problem is in cellular or non-cellular form.	String	TRUE	Cellular problem
			FALSE	Non-cellular problem
CtrFile	Ctr file for problem.	String	Filename	
VarFile	Var file for problem.	String	Filename	
ProblemType	Specifies whether an initial subproblem is to be used.	String	Full	No initial subproblem
			Sub	Use initial subproblem

If Cellular = “TRUE” then additional parameters must be given:

DemFile	Demand file for problem.	String	Filename
DefaultCosite	Specifies cosite constraint for all cells not explicitly given in demand file.	Integer	[1,INT_MAX]

If ProblemType = “Sub” then an additional parameter must be given to specify the file containing the initial subproblem.

LogFile	Log file specifying initial subproblem to be used.	String	Filename

Problem output

BestLog	Log file to store best subproblem found.	String	Filename	
OutputParameters	Only necessary if command line option –o is used. Specifies whether the parameters for each run are copied to the output file.	String	True	Output parameters
			False	No parameters output

If OutputParameters = “True” then each individual run can be restarted by copying the relevant parameters. However, this prevents the output being conveniently read into Matlab[19] for processing.

Algorithm parameters

NumberRuns	Number of individual runs to be performed.	Integer	[1,INT_MAX]	
BoundType	Specifies the algorithm to be used and the bounding method to evaluate each subproblem.	String	PTMP	Algorithm 1 with PTMP LP bound.
			PTMP_FAP	Algorithm 1 with PTMP FAP LP bound.
			PTMP_IP	Algorithm 1 with PTMP IP bound.
			PTMP_FAP_IP	Algorithm 1 with PTMP FAP IP bound.
			PTMP_EXT	Algorithm 2

[19] http://www.mathworks.com/

ObjectiveWeight	Weight in the cost function of the bound for the current subproblem.	Float	[-FLT_MAX,FLT_MAX]
SizeWeight	Weight in the cost function of the size of the current subproblem.	Float	[-FLT_MAX,FLT_MAX]

If BoundType = "PTMP_EXT" then additional parameters must be given:

Tolerance	Percentage above which the PTMP_FAP LP bound is applied.	Float	[0,100]

Simulated annealing

CoolingType	Cooling schedule to be used.	String	Linear	
			Geometric	
			Costa	
			LM	
			HS	
			Logarithmic	
DistributionType	Probability distribution to be used.	String	maxbolt	Maxwell-Boltzmann Distribution
			taylor	Taylor expanded Maxwell-Boltzmann Distribution
K	Scaling parameter for tunnelling frequency.	Float	[-FLT_MAX,FLT_MAX] \ {0}	

If CoolingType = "Logarithmic" then the following must also be set:

LogCClog	N_{log}	Integer	[1,INT_MAX]
LogCCgamma	N_{γ}	Float	[0,1]
LogCCwidth	width	Integer	[1,INT_MAX]
LogCCcold	cold	Integer	[1,INT_MAX]

For all other values of CoolingType, the following must be set:

Tstart	Starting temperature.	Float	[0,FLT_MAX]
Tfinish	Finishing temperature.	Float	[0,Tstart)
NFrozen	Number of failures before freeze occurs.	Integer	[1,INT_MAX]
NIterations	Number of iterations.	Integer	[1,INT_MAX]

together with the appropriate cooling constant:

LinearCC	Cooling constant, α_1.	Float	[0,FLT_MAX]
GeometricCC	Cooling constant, α_2.	Float	[0,1]
CostaCC	Cooling constant, α_3.	Float	[0,1]
LMCC	Cooling constant, α_4.	Float	[0,FLT_MAX]
HSCC	Cooling constant, α_5.	Float	[0,FLT_MAX]

Random seeds

Seed1	Random seed.	Integer	[0, 31328]
Seed2	Random seed.	Integer	[0, 30081]

Move generator

MoveType	Type of move used to generate new subproblems.	String	Random	Add/remove cells at random
			Connected	Add/remove cells only if connected to the current subproblem
MoveSize	Number of cells changed in each move.	Integer	[1,INT_MAX]	

Output

The progress of the algorithm is output to the screen and to a file if specified. Typical output is shown below:

```
% Run 0
%   iter    Bound SA-best SA-curr    size    temp         prob
       0        0       0       0       0       8            1 % B
% PTMP_FAP BOUND 0
       1        0       0       0       1       8            1 % T
% PTMP_FAP BOUND 0
       2        2       2       2       2       8            1 % B
% PTMP_FAP BOUND 2
       4        2       2       2       3       8            1 % T
% PTMP_FAP BOUND 2
      10        3       3       3       4       8            1 % B
% PTMP_FAP BOUND 3
      18        3       3       3       5       8            1 % T
% PTMP_FAP BOUND 4
      25        3       3       3       6       8            1 % T
% PTMP_FAP BOUND 4
      33        3       3       3       7       8            1 % T
% PTMP_FAP BOUND 5
      39        5       5       5       8       8            1 % B
% PTMP_FAP BOUND 5.5
      43        7       7       7       9       8            1 % B
% PTMP_FAP BOUND 7
      58        8       8       8      10       8            1 % B
% PTMP_FAP BOUND 9
% Bound 8

% +FAP 9
```

```
% Run 1
%   iter   Bound SA-best SA-curr    size    temp         prob
       0       0       0       0       0       8            1 % B
% PTMP_FAP BOUND 0
.
.
.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Best PTMP Bound 37 in run 2/50
% Best PTMP_FAP Bound 37 in run 2/50
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

The columns are as follows:

`iter`	The current iteration (number of subproblems tested)
`Bound`	The bound for the current subproblem
`SA-best`	The cost of the best subproblem so far.
`SA-curr`	The cost of the current subproblem
`size`	The size of the current subproblem
`temp`	The current temperature
`prob`	The probability of accepting the current subproblem
`% B`	This subproblem has the best cost so far
`% S`	This subproblem has a better cost than the last subproblem accepted
`% T`	A tunnel event has occurred

The line
% PTMP_FAP BOUND 6

only occurs for Algorithm 2 and shows that the PTMP bound of the current subproblem is large enough to evaluate the PTMP_FAP bound. The lines

```
% Bound 8

% +FAP 9
```

specify the best bounds for the current run.

Example parameter file

```
// hs parameter file example

// Problem input
Cellular      = "TRUE"           // Cellular problem
CtrFile       = "ph8.cell.ctr"
VarFile       = "ph8.cell.var"
ProblemType   = "Full"           // No initial subproblem used
DemFile       = "ph8.cell.dem"
DefaultCosite = 5

// Problem output
BestLog          = "best.log"    // To store subproblem with the best bound
OutputParameters = "True"        // Output parameters if an output file is
                                 // specified on the command line

// Algorithm parameters
NumberRuns       = 50
BoundType       = "PTMP_EXT"     // Use Algorithm 2
ObjectiveWeight = 1              // Cost function consists of bound only
SizeWeight      = 0
Tolerance       = 98             // Test all subproblems with PTMP above 98%
                                 // of best

// Simulated annealing
CoolingType       = "Logarithmic"
DistributionType  = "maxbolt"
K                 = 0.001
LogCClog          = 3000
LogCCgamma        = 0.1
LogCCwidth        = 2
LogCCcold         = 1

// Random seeds
Seed1 = 0
Seed2 = 0

// Move generator
MoveType = "Connected"
MoveSize = 1
```

ptmp – Calculation of bounds

Usage

```
ptmp –p parameter.par
```

Parameter settings

The settings to be specified in the parameter file are shown in the following tables, in the form:

Parameter name	Description	Type	Range of allowed values

Red parameters are compulsory in all cases.

Green parameters force additional parameters to be set.

Problem input

Cellular	Specifies whether the problem is in cellular or non-cellular form.	String	TRUE	Cellular problem
			FALSE	Non-cellular problem
CtrFile	Ctr file for problem.	String	Filename	
VarFile	Var file for problem.	String	Filename	
ProblemType	Specifies whether the bound is for a subproblem or a full problem.	String	Full	Full problem
			Sub	Subproblem

If Cellular = “TRUE” then additional parameters must be given:

DemFile	Demand file for problem.	String	Filename
DefaultCosite	Specifies cosite constraint for all cells not explicitly given in demand file.	Integer	[1,INT_MAX]

If ProblemType = “Sub” then an additional parameter must be given to specify the file containing the initial subproblem.

LogFile	Log file specifying subproblem to be used.	String	Filename

Problem output

LpFile	File to store linear program in CPLEX lp format.	String	Filename
SolnFile	File to store linear program solution	String	Filename

Bound parameters

BoundType	Bound to calculate.	String	PTMP	PTMP LP bound.
			PTMP_FAP	PTMP FAP LP bound.
			PTMP_IP	PTMP integer program.
			PTMP_FAP_IP	PTMP FAP integer program.

Output

The bound is output to the screen. The linear program is output to file in Cplex lp format and a solution is output to file.

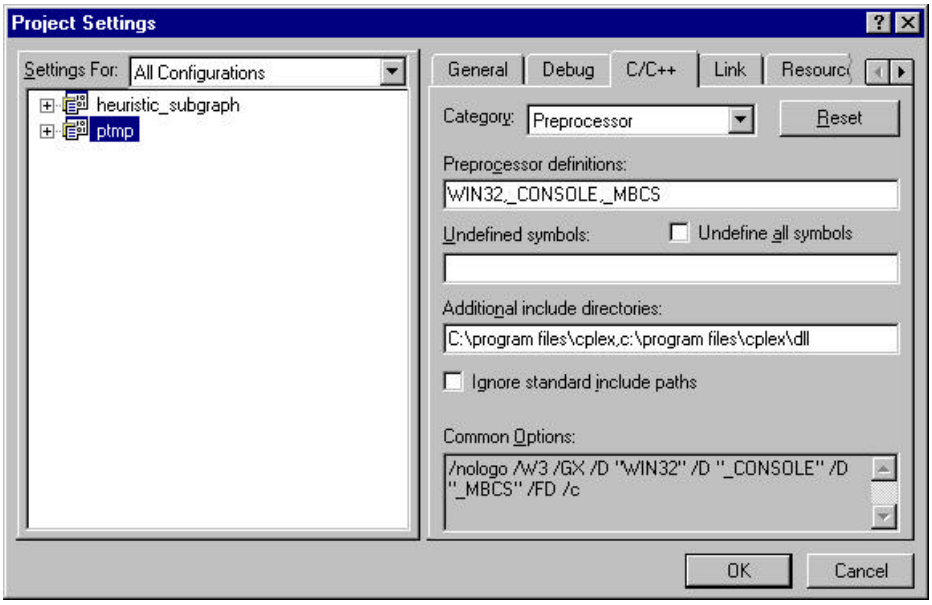
Example parameter file

```
// ptmp parameter file example

// Problem input
Cellular      = "TRUE"           // Cellular problem
CtrFile       = "ph8.cell.ctr"
VarFile       = "ph8.cell.var"
ProblemType   = "Sub"           // Calculate bound for a subproblem
LogFile       = "c0.cell.log"
DemFile       = "ph8.cell.dem"
DefaultCosite = 5

// Problem output
LpFile   = "c0.lp"
SolnFile = "soln.txt"

// Bound parameters
BoundType       = "PTMP_FAP"
```

Compilation details

The source code for `hs` and `ptmp` is written in C++. Workspace and project files are included for use with Microsoft Visual C++ 6.0. To successfully compile `hs` and `ptmp` the following steps must be taken:

1. Open the workspace `bounds.dsw`.
2. In `Project|Settings`, select the C/C++ tab, and in the Category preprocessor change the 'Additional include directories' to include the correct paths for the folders containing the cplex executable, and the cplex dll (this should be done for all configurations — debug and release).



3. Add the following Cplex files to the project:

```
calldll.c
calldll.h
cplex.h
dllcover.h
ntdll.h
```

4. The individual projects (ptmp and hs) can now be built.