1-1-2007

Virtual Backbone Configuration in Wireless Mesh Networks

Mike Morgan

Vic Grout
Glyndwr University, v.grout@glyndwr.ac.uk

Virtual Backbone Configuration in Wireless Mesh Networks

Abstract
This paper introduces methods for the minimisation of virtual backbone size in wireless mesh networks, subject to practical constraints. The methods are centralised, which limits their usage to static applications. Four algorithms are presented, one exact and three heuristic. The exact method guarantees to find an optimal solution but runs in exponential time. Of the three heuristics, one is shown to match the performance of the optimal algorithm for all problem instances tested. The problem is constrained to introduce potentially massive levels of redundancy into the network topology, making the designs survivable.

Keywords
Wireless networks, Backbone networks, Optimisation, Heuristics

Disciplines
Computer and Systems Architecture | Digital Communications and Networking | Hardware Systems | Systems and Communications

Comments
This paper was presented at the Third Collaborative Research Symposium on Security, E-Learning, Internet and Networking (SEIN 2007), 3rd International NRG Research Symposium, 14-15 June 2007, which was held by University of Plymouth and the symposium proceedings are available at http://www.cscan.org

Virtual Backbone Configuration in Wireless Mesh Networks

Mike Morgan and Vic Grout

Centre for Applied Internet Research (CAIR), NEWI, Wrexham, United Kingdom
e-mail: mi.morgan|v.grout@newi.ac.uk

Abstract

This paper introduces methods for the minimisation of virtual backbone size in wireless mesh networks, subject to practical constraints. The methods are centralised, which limits their usage to static applications. Four algorithms are presented, one exact and three heuristic. The exact method guarantees to find an optimal solution but runs in exponential time. Of the three heuristics, one is shown to match the performance of the optimal algorithm for all problem instances tested. The problem is constrained to introduce potentially massive levels of redundancy into the network topology, making the designs survivable.

Keywords

Wireless networks, Backbone networks, Optimisation, Heuristics

1. Introduction

The use of mesh topologies is a well-known approach to network design for both telephone trunk systems and Internet traffic. We introduce the technique by comparison to the point-to-multipoint (PMP) or star topology, which is more commonly used in wireless applications. Figure 1a shows a number of fixed locations which need to be connected wirelessly. In Figure 1b, we have a design connecting the locations in a PMP fashion. Three base stations are used, indicated by the black squares. These may be interconnected using fibre optic or higher capacity wireless links.

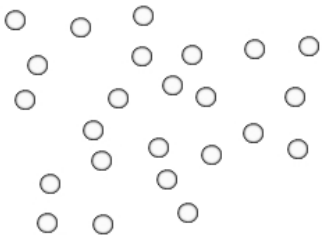

Figure 1a: Locations

Figure 1b: PMP Design

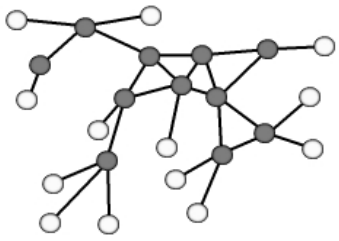

Figure 1c: Mesh Design

Figure 1c shows the mesh alternative. Here, the grey customer nodes are being used as distribution nodes or *relays*, which collectively form a *virtual backbone*. The relay nodes house multiple antennae and electronics for both transit traffic and drop-to-customer traffic, which is more costly than the equipment required at terminal nodes. As a result of this, the cost of installing a mesh network can grow more rapidly than for a PMP system, particularly in the approach where relay equipment is installed at all customer nodes (Anderson, 2003). However, this difficulty can be tackled by reducing the number of relays in the network. It is

clear from Figure 1c that not all nodes need have relay equipment installed. If the number of relays in the virtual backbone can be minimised, so can the cost of implementation.

A further problem with mesh networks is that distribution nodes may not be under the control of the network operator. If a customer disconnects his/her node, then other users may be affected - clearly not the case with a PMP system. This forms part of the motivation behind the approach whereby relay equipment is installed at every node, ignoring cost and maximising network redundancy. However, an alternative is to constrain the design in such a manner that redundant paths are maintained between distribution nodes and/or that terminal nodes have backup relays. Hence it would be advantageous if we could minimise the number of relays subject to connectivity constraints. A formal discussion follows in the next section.

2. Virtual Backbone Minimisation

We begin this section by outlining the unconstrained version of the problem, which is well known.

2.1 The Minimum Connected Dominating Set Problem (MCDS)

Define a problem graph to be $\boldsymbol{G} = (\boldsymbol{V},\boldsymbol{E})$ where $\boldsymbol{V}$ is a set of vertices representing locations to be connected and $\boldsymbol{E}$ is a set of edges representing potential wireless links between vertices in $\boldsymbol{V}$. Figure 2a gives an example.

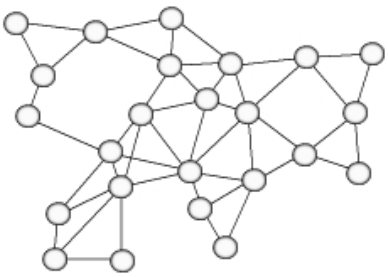

Figure 2a: A Problem Graph

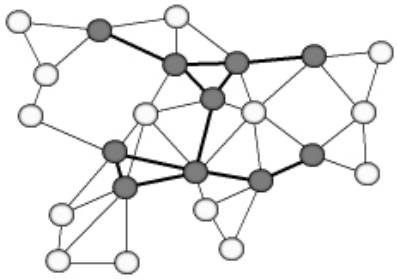

Figure 2b: A Connected Dominating Set

Part of our objective is to find a set of vertices $\boldsymbol{S} \subseteq \boldsymbol{V}$ such that $\boldsymbol{S}$ forms a connected component of $\boldsymbol{G}$ and each node in $\boldsymbol{V}$-$\boldsymbol{S}$ has a neighbour in $\boldsymbol{S}$. $\boldsymbol{S}$ represents the set of relays and is said to be *connected* and *dominating* if it meets these criteria (Figure 2b). $\boldsymbol{V}$-$\boldsymbol{S}$ is the set of terminal nodes and is said to be *dominated* by $\boldsymbol{S}$.

We wish to find a connected dominating set (CDS) of $\boldsymbol{G}$ with minimum size. The problem is known to graph theorists as the *minimum connected dominating set* (MCDS) problem and is NP-Complete (Garey and Johnson, 1979). Two heuristics for this problem are outlined by Morgan and Grout (2006) and shown to outperform established methods significantly.

2.2 The Minimum *k*-Connected *c*-Dominating Set Problem

The problem with MCDS designs such as in Figure 2b is one of reliability. The failure of a relay could not only leave terminals isolated, but also split the backbone into one or more fragments. Alternative designs are proposed in Figures 3a and 3b to resolve these issues.

Figure 3a: 2-Connected Dominating Set

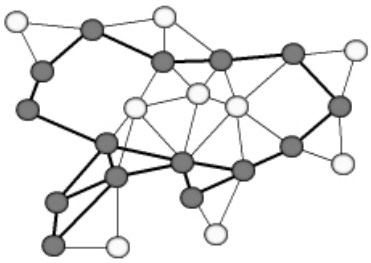

Figure 3b: 2-Connected 2-Dominating Set

A graph, or graph component is said to be *k-connected* if there are at least *k* vertex-independent paths between all pairs of vertices, or equivalently if *k* or more vertex deletions are required to disconnect it (Menger, 1927). In Figure 3a, the relay set ***S*** forms a 2-connected component of ***G***. This guarantees that the failure of a single relay cannot fragment the backbone. Similarly, a set ***S*** is said to be *c-dominating* if each vertex in ***V-S*** has at least *c* neighbours in ***S***. In Figure 3a some of the terminals still only have one relay as a neighbour and the failure of this relay could isolate the terminal. This situation is resolved with the 2-dominating example in Figure 3b.

With these definitions, we may now consider a more general form of the MCDS problem in which ***S*** is required to be *k*-connected and *c*-dominating. The virtual backbone must not only be connected, but must be able to survive at least *k*-1 node failures and terminals must have at least *c* adjacent relays. The ability to specify the *k*-connectivity of the backbone is particularly useful as networks may be designed to meet an estimated level of reliability with this approach (Kershenbaum, 1993).

However, it should be understood that the values of *k* and *c* may be limited by the connectivity of ***G***. If ***G*** is only 2-connected as in our example above, it becomes impossible to find a 3-connected 3-dominating set and unlikely that we will find a 3-connected 2-dominating one. We may, of course, constrain our problem so that we maintain ***G***'s level of connectivity if it is low, thus avoiding any significant loss in backbone reliability compared to the solution where all nodes are relays. In addition, it may be possible to increase connectivity by the introduction of additional non-customer, or *seed* nodes.

2.3 Additional Path Constraints

Whilst we have dealt with the issue of multiple paths to some extent in the previous subsection, it may be the case that certain vertex pairs have large quantities of high-priority traffic destined for one another and consequently that more multiple paths are required between them than between other nodes in the backbone. Alternatively, we may require that one node requires a greater number of disjoint paths to the remaining portion of the backbone (It may be an access point for an optical fibre network, for example). Examples are shown in Figures 4a and 4b.

In Figure 4a, we require that 3 vertex-disjoint paths are available between the vertices marked black. The result is that, although the failure of two nodes (or links) could fragment the backbone, there will still be a line of communication between this pair. Naturally, if multiple paths are required, it is essential that both nodes are relays. In Figure 4b, we have an example of an access network, where the square symbol is a portal to an optical fibre network. We

maintain the constraint that there must be 2 vertex-disjoint paths from all relays to the portal. The design is very similar to that in Figure 3a, with the alteration that the portal must be part of the backbone.

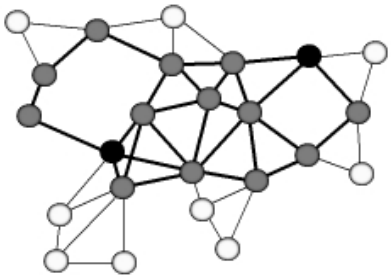

Figure 4a: Multiple Paths

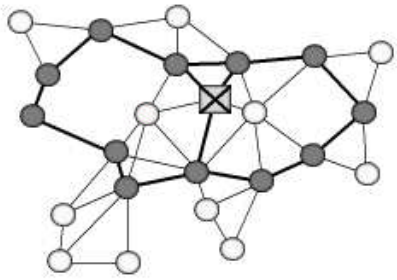

Figure 4b: Access Network

2.4 Node Constraints

The last two constraints concern nodes which must or cannot be relays. In the previous section, we find that specified access points and nodes with multiple path constraints must be relays. In addition to this, there are circumstances where relay equipment may already be installed at a location prior to design (in re-design problems, for example). Also, there may be circumstances where a given node may not be able to house relay equipment. These two types of constraint are simple to implement and may be specified as a design input. They will be referred to as *fixed* and *infeasible* relay/node constraints respectively, throughout the remainder of this text.

This section has extended the MCDS problem into a generalised, constrained form applicable to virtual backbone configuration. We will refer to it as the *Minimum Relay Problem* (MRP).

3. Exact and Heuristic Algorithms for MRP

In this section we introduce four algorithms for MRP, one exact and three heuristic. We begin with an exhaustive search technique which guarantees to find the optimum solution but runs in exponential time.

3.1 MRP/ES: An Exhaustive Search Algorithm for MRP

We may guarantee to find an optimal solution to the MRP as follows: Consider a problem graph $\boldsymbol{G = (V, E)}$ with n nodes and m edges. We may deduce that the number of possible MRP solutions is $O(2^n)$ as each of the n nodes may be in one of two states: relay or terminal. The presence of node constraints will reduce the search space to some degree.

For each of these possible solutions we need to evaluate whether or not all the problem constraints are satisfied. The complexity of this process will depend on the type of constraints present. A summary of complexities for each type of constraint is given in Table 1.

The c-domination constraint can be evaluated in two ways. Firstly, by scanning the edge lists of all terminal nodes to check for a minimum of c relays or secondly, by scanning the edge lists of all relays and incrementing a counter for each terminal found. The second method is efficient for small relay sets but the first can bail out as soon as it finds a non-dominated

terminal, so could be quicker in the event that many infeasible solutions are being tested. In either case the method has $O(m)$ time complexity.

Constraint	Complexity
c-domination	$O(m)$
k-connectivity*	$\begin{Bmatrix} O(m) \mid k < 3 \\ O(k^2 nm) \mid k \geq 3 \end{Bmatrix}$
p multiple paths	$O(pm)$ per constraint
node constraints	**n/a**

Table 1: Complexity of Constraint Evaluation

**Complexities for k-connectivity are based on methods used in this paper – lower complexities can be obtained for k=3 and above using more sophisticated techniques*

For k = 1 and 2, $O(m)$ algorithms may be used to test k–connectivity, namely breadth/depth first search for k=1 and the lowpoint algorithm (Tarjan, 1972) for k=2. An $O(m)$ algorithm for 3-connectivity does exist (Tarjan and Hopcroft, 1973), but its implementation is cumbersome. Similarly, efficient but logically complex algorithms could be used for k>3 (Matula, 1987) (Kanevsky and Ramachandran, 1991) but we have chosen to use the method proposed by Kleitman (1969) on account of its simplicity. As this method involves the evaluation of multiple vertex-disjoint paths, we will discuss path constraints before returning to it.

Path constraints may be evaluated using an adaptation of the Ford-Fulkerson algorithm (Ford and Fulkerson, 1956). This runs in $O(pm)$ time, where p is the number of vertex-disjoint paths required. In the event that multiple paths are required to an access point, we will need to test r-1 of these constraints, where r is the number of relays in the backbone. This increases the complexity to $O(pnm)$. Where k-connectivity ($k \geq 3$) is concerned, we use the following theorem (Kleitman, 1969):

> $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$ is k-connected if for any node $v \in \boldsymbol{V}$, there are k vertex-disjoint paths from v to each other node and the graph $\boldsymbol{G}'$, formed by removing v and all its incident edges from $\boldsymbol{G}$ is (k-1)-connected.

Thus k-connectivity can be tested with $O(kn)$ disjoint path calculations and the complexity of the entire process is $O(k^2nm)$. We must note that, with k-connectivity and multiple path calculations, we are only interested in testing the subgraph of $\boldsymbol{G}$ induced by $\boldsymbol{S}$. Therefore, algorithms used for testing these constraints are generally more efficient in practice than these complexities indicate, particularly when r is small.

We can deduce from Table 1 that the best complexity we could find for an evaluation of the entire search space would be $O(2^n m)$ and that this would be somewhat dependent on the value of k and the type of constraints present. Whilst we are aware of no technique that will improve on this bound, we can order the search in such a manner as to significantly reduce runtimes for the majority of cases. Figure 5 shows pseudocode for an algorithm that achieves this.

We define a *candidate vector* to be a boolean data structure containing a 1 for each relay node and a 0 for each terminal node. Fixed relays are assigned 1 and infeasible relays the value 0 for all candidate vectors. A feasible candidate is one which satisfies the problem's constraints.

```
procedure MRP/ES
begin
        initialise i ← 0
        repeat
                for all candidate vectors v_c with i (feasible, non-fixed) relays
                        if v_c is feasible
                                then return v_c          # v_c is the optimum
                i ← i+1
        until (all feasible nodes are relays)
        return ∅                                         # no feasible solution exists
end
```

Figure 5: Pseudocode for MRP/ES

Using this method we save evaluating all solutions with more than r_{opt} relays, where r_{opt} is number of non-fixed relays in the optimum solution. The only circumstance where we fail to reduce the running time is one in which no feasible solution exists. The number of candidates we need to evaluate is bounded above by:

$$\sum_{i=0}^{r_{opt}} \frac{n'!}{i!(n'-i)!} \tag{1}$$

Where n' is the number of non-constrained nodes. To illustrate the extent of this reduction, we will use a problem instance from the results section of this paper in which n=100 and r_{opt}=7. The search space size is 2^n or 1267650600228229401496703205376, however a maximum of only 17278988695 of these candidates need be evaluated to find the global optimum. Also, the size of the relay component is never greater than 7 and therefore its number of edges cannot exceed 42, greatly speeding up evaluation of the k-connectivity constraint.

Finally, we may decrease the search space further by considering the value of k and the connectivity of $\boldsymbol{G}$. For example, if we require 1-connectivity we may assume that all separation vertices of $\boldsymbol{G}$ (vertices whose removal would disconnect the graph) must be relays. Similarly, if 2-connectivity is required, all separation pairs of $\boldsymbol{G}$ must be relays. Likewise for 3-connectivity and separation triples etc. Constraining these nodes will improve the efficiency of any of the algorithms introduced, but the effect is most significant with exhaustive search.

Although we have increased the efficiency of the search process immensely, the exhaustive search algorithm is still only appropriate for small, dense problem instances. Some of the examples cited in the results section of this paper took up to several days to solve. Runtimes are also unpredictable as the value of r_{opt} is not known at the outset, although upper bounds can be placed on r_{opt} using the heuristics outlined in the following subsections.

3.2 MRP/Drop and MRP/GRASP: Construction Heuristics for MRP

The first heuristic in this section is a greedy algorithm called MRP/Drop. The second is a semi-greedy adaptation of MRP/Drop employing the GRASP metaheuristic and is consequently named MRP/GRASP. A description of MRP/Drop is given in Figure 6.

We define the *relay-degree* of a node to be the number of relays adjacent to it. MRP/Drop starts with a full set of relays, so far as constraints will allow and repeatedly tries to drop the relay with smallest relay-degree. If the resulting vector is feasible, the relay remains dropped.

Otherwise it is reinstated and marked un-droppable. Fixed relays are marked un-droppable at the outset. If there are several relays with equal-smallest relay-degree, u is chosen at random from among them. The algorithm terminates when all the original relays have either become terminals or have been marked.

```
procedure MRP/Drop
begin
        initialise candidate vector v_c so that all feasible nodes are relays
        mark all fixed relays
        repeat
                select a non-marked relay u from v_c with minimum relay-degree
                make u a terminal
                if v_c is infeasible
                        then make u a relay again and mark it
        until (all relays are marked)
end
```

Figure 6: Pseudocode for MRP/Drop

The complexity of this algorithm will be defined in terms of the number of candidates which need to be tested, understanding that the complexity of each test depends upon the type of constraints present. MRP/Drop evaluates a maximum n' candidates, massively reducing the time complexity from (1). However, there is no guarantee that we will find the optimum solution and greedy algorithms are generally outperformed by more sophisticated heuristics.

```
procedure MRP/GRASP
begin
        initialise candidate vector v_best so that all feasible nodes are relays
        repeat
                initialise candidate vector v_c so that all feasible nodes are relays
                mark all fixed relays
                repeat
                        randomly select a non-marked relay u from v_c
                        (such that u's relay-degree is within t of minimum)
                        make u a terminal
                        if v_c is infeasible
                                then make u a relay again and mark it
                until (all relays are marked)
                if v_c contains fewer relays than v_best
                        then v_best ← v_c
        until (no improvement found in v_best for I_max iterations)
end
```

Figure 7: Pseudocode for MRP/GRASP

Fortunately, it is very easy to modify this algorithm into a *Greedy Random Adaptive Search Procedure* (GRASP). Recall that in MRP/Drop, we choose u at random from relays with minimum relay-degree. To convert to a GRASP algorithm, we specify a *threshold* (t) so that u is now chosen at random from relays with relay-degree within t of the minimum value. The procedure is then repeated until no improvement has been found in the last I_{max} iterations. Pseudocode for MRP/GRASP is found in Figure 7.

Further improvements can be made in efficiency if we consider that evaluation of k-connectivity and multiple path constraints grows more time consuming with increasing r. Therefore, beginning with all feasible nodes as relays may slow the algorithm considerably. An alternative is to begin with a smaller, feasible solution. Figure 8 shows a randomised

algorithm which creates smaller feasible solutions in most cases and may be used as an alternative means of initialising v_c for both MRP/Drop and MRP/GRASP. We start with a small probability p_{rel} of a non-constrained node becoming a relay and increase it by p_{inc} until we get a feasible solution, thus avoiding constraint evaluations where r is excessively large during the drop process.

```
procedure RandomInitialise
begin
        initialise p_rel, p_inc and v_c
        repeat
                for all non-constrained nodes u in G
                        if random[0,1)< p_rel
                                then make u a relay in v_c
                                else make u a terminal v_c
                p_rel ← p_rel + p_inc
        until (v_c is feasible or p_rel>1)
end
```

Figure 8: Pseudocode for MRP/RandomInitialise

The GRASP algorithm gives improvements over Drop but can become time consuming with complex constraints. It is also commonly accepted that construction methods like MRP/Drop and MRP/GRASP do not perform particularly well without a local search procedure to improve on candidate solutions. Therefore, it behoves us to present such a method in the next subsection.

3.3 MRP/SA: A Simulated Annealing Algorithm for MRP

Simulated Annealing (Aarts and Korst, 1989) is a stochastic variant on local search which has proved successful for the unconstrained MCDS problem (Morgan and Grout, 2006). Pseudocode for a generic simulated annealing algorithm is given in Figure 9.

```
procedure simulated annealing
begin
        initialize temperature T
        initialise v_c
        repeat
                repeat
                        select a new candidate v_n in the neighbourhood of v_c
                        if eval(v_n)<eval(v_c)
                                then v_c ← v_n
                        else if random[0,1) < e^((eval(v_c)−eval(v_n))/T)
                                then v_c ← v_n
                until (termination-condtion)
                T ← αT
        until (halting-criterion)
end
```

Figure 9: Pseudocode for a Generic SA Algorithm

The algorithm starts with a candidate vector v_c and makes some small alteration to it to create a new vector v_n: a process often referred to as a *transition* or *move.* This new vector is said to be in the *search neighbourhood* of v_c. If v_n is found to be an improvement on v_c by the evaluation function eval(), then v_c is replaced by v_n. If v_n is not found to be an improvement there is still a possibility it will replace v_c. However, the probability of a non-improving

solution being accepted is reduced periodically by reducing the temperature variable *T*. Here, *T* is multiplied by a value α $(0 \leq \alpha < 1)$ each time the termination condition is met.

A simple evaluation function for MRP would return *r* for feasible solutions and infinity otherwise. However, we have improved upon this using the intuition that two solutions with the same *r* may not be equally desirable and that the solution whose relays have the greatest total degree ought to be preferred. The greater degree solution can be considered to cover the nodes more thoroughly and give us a better chance of dropping a relay without violating a constraint. So the evaluation function returns $2mr-C$ for feasible solutions where *C* is the total degree of the relays (the value of *C* cannot exceed $2m$: the total degree of the problem graph – hence the multiplying factor). Infinity is returned for infeasible solutions.

The search neighbourhood is made up of solutions obtainable by adding and/or dropping 1 node to/from the relay set. Alterations are selected at random, whereby a probability p_{move} is defined to be the probability of not adding and not dropping each time. Note that a move in which a terminal is added and no relay dropped cannot improve eval(v_n) over eval(v_c) but we can exploit SA's probabilistic acceptance of non-improving solutions to increase *r* temporarily when such a move is performed. This helps to diversify the search.

In the interests of efficiency we have organised moves so that relay nodes are dropped in an outer loop and added in an inner one. This is because multiple path constraints can be evaluated incrementally on adding, reducing the complexity of moves in the inner loop. To cover the event that a non-dropping move is selected, we maintain data for the candidate before the relay was dropped as well as after and add to the appropriate vector (Figure 11).

Incremental evaluation of multiple path constraints works on the basis that we cannot reduce the number of vertex-disjoint paths between a node pair by more than 1 if only 1 node has been removed. Suppose *p* paths are required between two relays *A* and *B*. On dropping a relay *C* from the relay component we then run the modified Ford-Fulkerson algorithm using *A* as the source and *B* as the destination, to see if the constraint has been violated. If it hasn't, we certainly won't violate it when we add a node back later, so no checking is necessary in the inner loop of our algorithm. If it has, we know that one or more minimum vertex-cuts of size p-1 have been created between *A* and *B*. the algorithm will also have return the set *X* of nodes on *A*'s side of the first such cut. We may now mark the constraint as violated and run the algorithm again using *B* as the source and *A* as the destination. Naturally we will get the same answer, but we will also obtain the set *X'* of nodes on *B*'s side of the last minimum cut. On adding a new relay *D*, we may conclude that the constraint is satisfied if and only if *D* has one or more neighbours in *X* and one or more neighbours in *X'* – or in other words if *D* creates a new path between the two sets. The process in the inner loop requires $O(\Delta)$ time per path constraint where Δ is the highest degree of all nodes in ***G***. Figure 10 gives an illustration where p=3. Most importantly, this process is used to speed up k-connectivity tests where k>2.

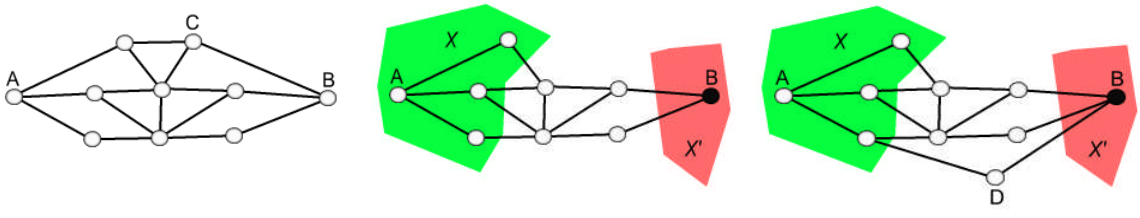

Figure 10: Incremental Evaluation of Path Constraints

For each relay dropped, we produce a *candidate list* of terminal nodes to add to the relay set. When a relay has been dropped, the domination of its terminal neighbours is reduced. So the candidate list is made up of the nodes which dominate most of these terminals, these can be easily found as we know they will be 2-hop neighbours of the dropped relay. If we cannot find enough eligible two-hop neighbours, the remaining portion of the list is filled with random terminal selections from ***G***. The process requires $O(\Delta^2)$ time. Pseudocode for the MRP/SA algorithm is given in Figure 11.

```
procedure MRP/SA
begin
    initialize temperature T, v_c, v_best
    repeat
        for all relays i
            drop i from v_c to obtain v_c'
            generate candidate list
            for all terminals j in candidate list
                randomly select type of move (add/drop/swap)
                if add
                    then add j to v_c to obtain v_n
                else if drop
                    then v_n ← v_c'
                else if swap
                    then add j to v_c' to obtain v_n
                if eval(v_n)<eval(v_c)
                    then v_c ← v_n
                else if random[0,1) < e^((eval(v_c)−eval(v_n))/T)
                    then v_c ← v_n
                if eval(v_c)<eval(v_best)
                    then v_c ← v_best
            T ← αT
    until (halting-criterion)
end
```

Figure 11: Pseudocode for MRP/SA

The halting criterion used is that there must have been no change in eval(v_c) over one iteration of the outer loop. In addition, we specify a minimum number of iterations $I_{\min}$ to prevent premature convergence of the algorithm.

4. Results

Problem instances were generated by the specification of three graph parameters. These were the number of nodes *n*, the maximum transmission distance MTD $(0 < MTD \leq 1)$ and a line-of-sight probability LOS $(0 < LOS \leq 1)$. The nodes were placed randomly on a unit square. Edges were created between nodes if the distance between them was less than *MTD* and *LOS* was greater than a random value between 0 and 1. Two probabilities p_{fixed} and p_{inf} were defined so that each node had a p_{fixed} chance of being a fixed relay and a p_{inf} chance of being an infeasible one. Multiple paths were also selected at random, but in this case the number of constraints n_{path} was specified precisely, with source and destination vertices being chosen at random for each of the n_{path} constraints. The number of paths *p* also had to be specified and for the purpose of these tests was kept constant for all constrained pairs. Parameters used for

MRP/SA were T=100, α=0.97, p_{move}=0.3, I_{min}=40 and a candidate list length of 20. For MRP/GRASP, t=3 and $I_{max}=n/4$. MRP/RandomInitialise was used for both MRP/GRASP and MRP/Drop, with p_{rel}=0.001 and p_{inc}=0.003.

4.1 Comparison Between Exhaustive Search and Simulated Annealing

First of all we turn to those instances for which the optimum solution is known. Unfortunately there are few of these as the MRP/ES algorithm runs very slowly. The results we have are also restricted to values of k<3, c=1 and no further constraints defined (except that fixed relay constraints were used to eliminate separation vertices as discussed in section 3.1). The SA results were compared with MRP/Drop which was used to initialise v_c for SA throughout this section. Tables 2 and 3 give a summary of results for k=1 and k=2. Observe that MRP/SA finds the optimum solution for all instances.

n	***MTD***	***LOS***	***MRP/ES***	***MRP/SA***	***MRP/Drop***
100	0.5	0.5	7	7	9
90	0.6	0.5	4	4	6
80	0.6	0.6	4	4	6
80	0.6	0.5	5	5	8
80	0.5	0.5	6	6	8
70	0.5	0.5	6	6	8
60	0.6	0.5	5	5	7
60	0.5	0.5	5	5	7
60	0.5	0.4	7	7	8
50	0.5	0.4	8	8	9
50	0.4	0.4	11	11	13
40	0.4	0.4	7	7	7
35	0.4	0.4	9	9	9
30	0.4	0.6	8	8	9
30	0.4	0.5	9	9	10
30	0.4	0.4	8	8	9
25	0.4	0.6	8	8	8
Mean Excess				**0%**	**20.5%**

Table 2: Optimality Tests for *k*=1

n	***MTD***	***LOS***	***MRP/ES***	***MRP/SA***	***MRP/Drop***
50	0.5	0.4	8	8	9
45	0.5	0.4	8	8	11
40	0.5	0.4	5	5	5
35	0.5	0.4	7	7	10
50	0.5	0.5	6	6	6
40	0.5	0.5	7	7	7
Mean Excess				**0%**	**17.1%**

Table 3: Optimality Tests for *k*=2

4.2 Comparisons between MRP/SA, MRP/Drop and MRP/GRASP

In the absence of a known optimum for more complex instances, we look for a relative improvement in CDS size for SA and GRASP over the greedy MRP/Drop. We express these as percentages in the tables that follow. Table 4 shows some results for 2-connected backbones with n in the range $200 \le n \le 1000$.

	MEAN CDS SIZES			MEAN RUNTIMES (s)			Improvement (%)	
n	***Drop***	***SA***	***GRASP***	***Drop***	***SA***	***GRASP***	***SA***	***GRASP***
200	43.7	32.85	38.15	0.23	3.8	10.4	24.83	12.7
300	47.7	34.1	41.95	0.37	5.68	16.85	28.51	12.05
400	50.25	35.45	44.3	0.53	7.53	25.9	29.45	11.84
500	52.7	35.65	46.7	0.84	8.84	36.77	32.35	11.39
600	53.95	36.7	48.25	1.25	12.31	62.26	31.97	10.57
700	55.85	38.15	50.45	1.66	14.05	84.49	31.69	9.67
800	57.95	39.05	51.35	2.4	19.33	115.78	32.61	11.39
900	58.25	40.35	52.6	3.31	21.94	168.02	30.73	9.7
1000	58.7	40.45	53.9	4.23	26.3	210.73	31.09	8.18
Mean Improvement over MRP/Drop (%)							**30.36**	**10.83**

Table 4: Mean CDS Size Improvements and Runtimes for Variable *n* (20 runs)
k=2, c=1, MTD=0.2, LOS=0.5, $p_{fixed}=p_{inf}=n_{path}=0$

The Improvement for SA over Drop is greater than that found for the optimal examples in Table 3 and the runtimes are also preferable to GRASP. A comparison of performances for a variety of values of *k*-connectivity and *c*-domination is given in Table 5.

		MEAN CDS SIZES			MEAN RUNTIMES (s)			Improvement (%)	
k	***c***	***Drop***	***SA***	***GRASP***	***Drop***	***SA***	***GRASP***	***SA***	***GRASP***
1	**1**	14.6	10.4	12.45	0.07	1.27	3.43	28.77	14.73
2	**1**	17.35	11.4	14.05	0.11	1.48	5.11	34.29	19.02
2	**2**	23.2	17.9	20.4	0.11	2.16	4.5	22.84	12.07
3	**1**	23.1	16.65	18.85	0.57	5.09	26.95	27.92	18.4
3	**2**	26.9	20.7	23.5	0.52	6.79	26.43	23.05	12.64
4	**1**	29.85	21.75	22.8	1.33	12.22	73.18	27.14	23.62
4	**2**	30.35	24.4	26.75	1.42	14.68	77.81	19.6	11.86
5	**1**	33.6	26.8	27.3	3.39	27.46	200.87	20.24	18.75
5	**2**	35.8	29.25	31.2	3.71	34.1	193.05	18.3	12.85
Mean Improvement over MRP/Drop (%)								**23.55**	**15.01**

Table 5: Mean CDS Size Improvements and Runtimes for Variable *k* and *c* (20 runs)
n=200, MTD=0.3, LOS=0.8, $p_{fixed}=p_{inf}=n_{path}=0$

SA still performs best here but becomes less successful with increasing connectivity. Runtimes increase significantly for all algorithms but are small enough to be acceptable. The algorithms' constraint handling capabilities are tested in the next three tables.

	MEAN CDS SIZES			MEAN RUNTIMES (s)			Improvement (%)	
npath	***Drop***	***SA***	***GRASP***	***Drop***	***SA***	***GRASP***	***SA***	***GRASP***
0	53.12	36.18	46.78	0.71	9.29	36.26	31.89	11.94
2	56.72	39.22	50.84	0.99	12.93	48.89	30.85	10.37
4	60.02	41.92	53.38	1.15	13.62	55.89	30.16	11.06
6	61.96	44.72	55.82	1.13	15.67	63.12	27.82	9.91
8	64.38	47.72	58.58	1.35	17.46	65.67	25.88	9.01
10	66.5	50.12	60.62	1.47	19.07	73.24	24.63	8.84
12	69.14	52.9	62.5	1.63	20.12	77.49	23.49	9.6
14	70	55.16	64.18	1.75	21.52	80.76	21.2	8.31
16	71.62	58.04	66.26	1.89	23.02	96.12	18.96	7.48
18	74.86	60.96	69.22	1.88	24.16	99.32	18.57	7.53
20	75.68	63.3	70.8	2.02	24.42	99.28	16.36	6.45
Mean Improvement over MRP/Drop (%)							**24.53**	**9.14**

Table 6: Mean CDS Size Improvements and Runtimes for Variable n_{path} (50 runs)
k=2, c=1, n=500, MTD=0.2, LOS=0.5, $p_{fixed}=p_{inf}=0$*, p=3*

Table 6 shows a deterioration in SA performance comparable with that for connectivity constraints, whereas GRASP shows less ability to cope with multiple paths. Tables 7 and 8 show the effect of increasing *pfixed* and *pinf* on both algorithms.

	MEAN CDS SIZES			MEAN RUNTIMES (s)			Improvement (%)	
p_{fixed}	*Drop*	*SA*	*GRASP*	*Drop*	*SA*	*GRASP*	*SA*	*GRASP*
0	32.12	22.84	27.44	0.17	2.67	7.98	28.89	14.57
1	33.56	24	28.42	0.16	2.63	7.72	28.49	15.32
2	33.7	24.96	29.5	0.16	2.64	7.88	25.93	12.46
3	35.02	26.2	30.38	0.16	2.57	7.63	25.19	13.25
4	35.62	26.62	30.82	0.15	2.41	7.46	25.27	13.48
5	37.16	28.74	32.28	0.15	2.34	7.15	22.66	13.13
6	37.82	29.6	32.92	0.15	2.23	7.26	21.73	12.96
7	39.12	31.14	34.5	0.15	2.29	6.97	20.4	11.81
8	39.36	32.16	34.98	0.14	2.11	6.59	18.29	11.13
9	40.32	33.48	36.18	0.15	2.11	6.41	16.96	10.27
10	41.82	34.46	36.94	0.14	2.16	6.64	17.6	11.67
Mean Improvement over MRP/Drop (%)							**22.86**	**12.73**

Table 7: Mean CDS Size Improvements and Runtimes for Variable p_{fixed} (20 runs)
k=2, c=1, n=500, MTD=0.2, LOS=0.5, n_{path}=p_{inf} =0

Again, the performance of MRP/SA drops off for both type of constraint, but perhaps the most remarkable feature of Table 8 is the consistency of MRP/Drop.

	MEAN CDS SIZES			MEAN RUNTIMES (s)			Improvement (%)	
p_{inf}	*Drop*	*SA*	*GRASP*	*Drop*	*SA*	*GRASP*	*SA*	*GRASP*
0	33.95	22.6	27.15	0.15	2.57	7.51	33.43	20.03
5	32.2	22.55	27.15	0.16	2.29	8.14	29.97	15.68
10	32.55	23	27.05	0.18	2.06	8.45	29.34	16.9
15	33.8	23.6	27.35	0.17	1.97	8.46	30.18	19.08
20	31.15	23.7	27.6	0.18	1.96	9.37	23.92	11.4
25	32.65	24.9	28.05	0.17	1.76	9.37	23.74	14.09
30	32.35	25.2	27.8	0.18	1.64	8.12	22.1	14.06
35	33.9	25.85	28.45	0.19	1.59	9.58	23.75	16.08
40	33.35	26.75	28.25	0.2	1.41	9.78	19.79	15.29
45	32.95	26.9	28.05	0.19	1.39	9.97	18.36	14.87
50	32.65	27.15	29.05	0.19	1.35	9.25	16.85	11.03
Mean Improvement over MRP/Drop (%)							**24.67**	**15.32**

Table 8: Mean CDS Size Improvements and Runtimes for Variable p_{inf} (20 runs)
k=2, c=1, n=200, MTD=0.2, LOS=0.8, n_{path}=p_{fixed} =0

5. Conclusions and Future Work

Of the algorithms presented, it appears that MRP/SA is a powerful algorithm for mesh backbone optimisation. It produces results comparable with exhaustive search for cases small enough for that process to be an option and outperforms both the construction algorithms presented. However, MRP/SA does require a construction algorithm to initialise its candidate vector and MRP/Drop and MRP/GRASP can both be considered options in this case. If MRP/GRASP is used, multiple MRP/SA passes can be applied to elite or to all candidates produced during the construction process. This would create a far more time consuming algorithm but given that may be tolerable for design problems.

Prospects for future work include variations on the objective function outlined here. The process of simply minimising relays might be exchanged for one in which the degree of relays is minimised. It is mentioned in section 1 that part of the motive for relay minimisation is the fact that multiple antennae are present on the relay nodes. A degree minimisation would be the process of minimising the number of such antennae, rather than the number of relays but it could be more likely that some weighted combination of the two would be required for practical problems. This would complicate the search process as not only relays but links adjacent to relays would need to be specified as part of the solution data structure, increasing the search space to 2^m. The growth in search space is massive considering the upper bound on m is n^2. Also, the link specification would mean that links, rather than nodes might need to be swapped in and out for MRP/SA. This would, no doubt increase the complexity of that algorithm.

A start has been made on the minimum relay problem, but further investigation and implementation is necessary before we can claim a complete solution, even heuristically. The focus of research from this point onwards will be on increasing the scope of the problem definition and producing/modifying algorithms to solve it.

6. References

Aarts, E and Korst, J. (1989) *Simulated Annealing and Boltzmann Machines*, Wiley

Anderson, H. (2003), *Fixed Broadband Wireless System Design*, Wiley, Chichester.

Ford, L. and Fulkerson, D. (1956), "Maximal Flow through a Network", *Canadian Journal of Mathematics*, 8, pp399-404.

Garey, M. and Johnson, D. (1979), *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, New York.

Hopcroft, J. and Tarjan, R. (1973) "Dividing a Graph into Triconnected Components", *SIAM journal of Computing*, Vol.2, No.3, pp135-158.

Kanevsky, A. and Ramachandran, V. (1991) "Improved Algorithms for Graph Four-Connectivity", *Journal of Computer System Sciences*, 42, pp288-306

Kershenbaum, A. (1993), *Telecommunications Network Design Algorithms*, McGraw-Hill

Kleitman, D. (1969), "Methods of Investigating Connectivity of Large Graphs", *IEEE Transactions on Circuit Theory (Corresp.)*, CT-16, pp232-233.

Matula, D. (1987), "Determining edge connectivity in O(nm)", *Proceedings of the 28th Annual Symposium on Foundations of Computer Science, 12-14 October 1987,* Los Angeles, California, pp249-251

Menger, K. (1927), "Zur allgemeinen Kurventheorie", *Fundamenta Mathematicae*, Vol. 10, pp96-115.

Morgan, M. and Grout, V. (2006), "Optimisation Techniques for Wireless Networks", *Proceedings of the Sixth International Network Conference: INC 2006,* Plymouth, pp 339-346.

Tarjan R. (1972), "Depth Search and Linear Graph Algorithms" *SIAM Journal of Computing,* Vol.1, No. 2, pp146-160.