7-1-2007

Practical Large-Scale Network Design with Variable Costs for Links and Switches

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

Stuart Cunningham
Glyndwr University, s.cunningham@glyndwr.ac.uk

Rich Picking
Glyndwr University, Wrexham, r.picking@glyndwr.ac.uk

Practical Large-Scale Network Design with Variable Costs for Links and Switches

Abstract
This paper considers communication network design problems that arise in the real world, with large numbers of nodes and link and switch costs dependent upon their traffic capacity. Such costs, in turn, depend upon network topology so are not fixed at the start of, or through, any optimisation process. Realistic topological restrictions are also discussed. The limitations of conventional approaches – both constructive and search based – are noted and the requirements of practical optimisation methods explored. Two workable approaches to network design - one an established local search variant, another a more novel geometric approach - are introduced. Five different algorithms, ranging from exhaustive search to fast heuristic are compared with experimental results given in conclusion.

Keywords
Algorithms and heuristics, Large-scale network design, Optimisation, Variable costs

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

Practical Large-Scale Network Design with Variable Costs for Links and Switches

Vic Grout, Stuart Cunningham and Rich Picking
v.grout@newi.ac.uk s.cunningham@newi.ac.uk r.picking@newi.ac.uk
University of Wales, NEWI, Wrexham, North Wales, UK

Summary
This paper considers communication network design problems that arise in the real world, with large numbers of nodes and link and switch costs dependent upon their traffic capacity. Such costs, in turn, depend upon network topology so are not fixed at the start of, or through, any optimisation process. Realistic topological restrictions are also discussed. The limitations of conventional approaches – both constructive and search based – are noted and the requirements of practical optimisation methods explored. Two workable approaches to network design - one an established local search variant, another a more novel geometric approach - are introduced. Five different algorithms, ranging from exhaustive search to fast heuristic are compared with experimental results given in conclusion.

Key words:
Algorithms and heuristics, Large-scale network design, Optimisation, Variable costs

1. Introduction

The topological network design problem (TNDP) for fixed networks in general and communication networks in particular is often considered well-solved. In the standard formulation, n nodes are to be interconnected with c_{ij} representing the cost of connecting node i directly to node j. The problem is then to find a connecting set of links minimising

$$C^* = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} c_{ij} \ . \qquad (1)$$

Early constructive algorithms (e.g. [1]) solve the problem to optimality in its unconstrained form and produce solutions of the type shown in Fig. 1. Capacity constraints can be applied although the problem then becomes NP-hard [2] and the necessarily adapted heuristics (e.g. [3]) only yield approximate solutions. However, with an initial solution in place, various classes of local search heuristics such as tabu-search [4], simulated annealing [5] or genetic algorithms [6] can be applied to perturbate parts of the solution to look for improvement. Ant Colony algorithms [7] also focus on static problems such as this. A common, generic, but as we shall see - unrealistic, approach is to formulate the problem in Linear or Integer Programming [8].

This paper begins by outlining the shortcomings of this formulation of the TNDP and its associated algorithmic solutions in practical applications. It then discusses the requirements of a real-world fixed network design optimisation process and introduces and compares various *effective* solutions. The terms *node*, *switch* and *traffic* are used freely in the text but may be substituted by/for any appropriate equivalent in whatever underlying communications technology is being considered.

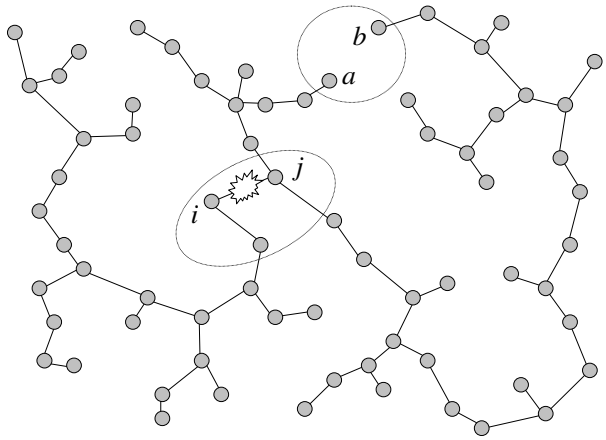

Fig. 1. Minimal Spanning Tree (MST) solution. The network has long paths between nodes (e.g. *a* & *b*) and is extremely vulnerable to failure. A failure of node *i* or link *(i, j)*, for example, splits the network in two.

2. Problems with the TNDP Formulation

The TNDP solution shown in Fig. 1. is a Minimal Spanning Tree (MST). There are two clear objections to such a topology:

- The connecting network has long, inefficient paths (in terms of number of links), even between geographically close nodes.

Manuscript received July 20, 2007
Manuscript revised July 25, 2007

- The solution network is extremely vulnerable to component failure. There is no redundancy. A fault at any node or link disconnects the network.

However, there are also difficulties associated with this simplistic notion of cost, independent of the method of solution:

- Only link (e.g. transmission) costs are considered; node (e.g. switching) costs are ignored.
- Traffic will flow, possibly asymmetrically, in both directions on a link; the structure of link costs should reflect this.
- Costs are taken as fixed in the statement of the problem and throughout any optimisation process, irrespective of network topology.

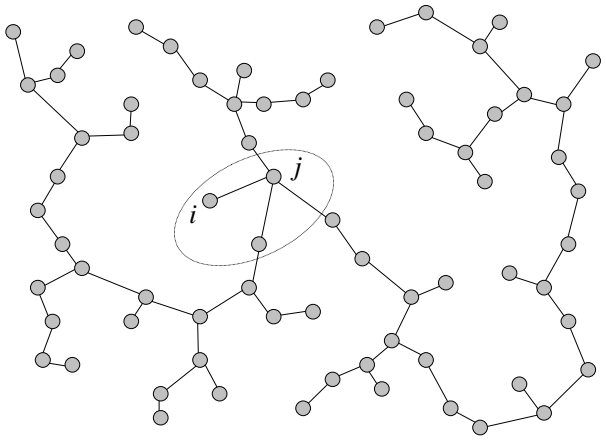

Fig. 2. A variant of Fig. 1. The link *(i, j)* carries only local traffic and is of small capacity – so will cost less. Also, *i* is no longer an expensive switch.

The final point warrants further explanation. The true cost of a link will depend partly on its length (which is known in advance) but also on its capacity - the level of traffic it can handle (which is not). In Fig. 1, the link *(i, j)* carries traffic from one side of the network to the other and will be of high capacity. In Fig 2, it carries local traffic only and will be much smaller. A similar variance applies to switches. The cost of a link or switch consequently depends on the solution topology - so cannot be fixed for the duration of the optimisation process.

For large numbers of nodes a more realistic network topology (for the same set of nodes as Fig. 2) is shown in Fig. 3. A subset of nodes (switches) has been chosen to concentrate and relay traffic among the remainder through a mesh or partial-mesh core network. The maximum path length between any node pair is reduced significantly and there is some tolerance of failure, at least in the core network, provided by redundant links. We take this as our model for a practical network design in this paper. Other forms are possible of course, such as:

- Constrained full-mesh or star core networks,
- Multiply-connected (non-switch) nodes,
- Multi-level (>2) networks.

The techniques discussed in this paper extend without difficulty to these variants. However, for brevity, the formulation of the problem here is based on the topology in Fig. 3. To generalize would extend the paper unnecessarily.

3. A Practical Formulation

Link costs remain variable, however, and switch costs should be considered. If we adopt the convention of using uppercase characters for switches and lowercase for non-switches then, in general, $c_s(i)=0$, $c_s(X)>0$, $c_l(i,j)=0$ and, where the link in question is present, $c_l(X,Y)>c_l(i,X)>0$, where c_s and c_l are the costs of switches and links respectively. More precisely, if a link L carries traffic t over a distance d then $c_l(L) = f_l(t,d)$. If a switch S processes traffic T then $c_s(S) = f_s(T)$. f_l and f_s may be any well-defined functions, dependent upon the underlying technology, and are not considered further until Section 7. Define t_{ij} to be the traffic between end-points i and j, that is, the traffic originating at i and destined for j. Define d_{ij} to be the 'distance' between i and j. This may the Euclidean straight line ($d_{ij} = [(x_i-x_j)^2+(y_i-y_j)^2]^{½}$ where (x_i,y_i) and (x_j,y_j) are the Cartesian coordinates of i and j) or weighted to reflect local factors. If a link is infeasible then $d_{ij}=\infty$. The cost of a link from a non-switch i to its *parent* switch X is then given by

$$c_l(i,X) = f_l(\sum_{j=1}^{n} t_{ij}, d_{iX}) \qquad (2)$$

with a corresponding cost $c(X, i)$ in the other direction. Define Γ_X to be the set of nodes with X as their parent in a given configuration/solution. Then the cost of the switch X is given by

$$c_s(X) = f_s(\sum_{i\in\Gamma_X}\sum_{j=1}^{n}(t_{ij}+t_{ji})) . \qquad (3)$$

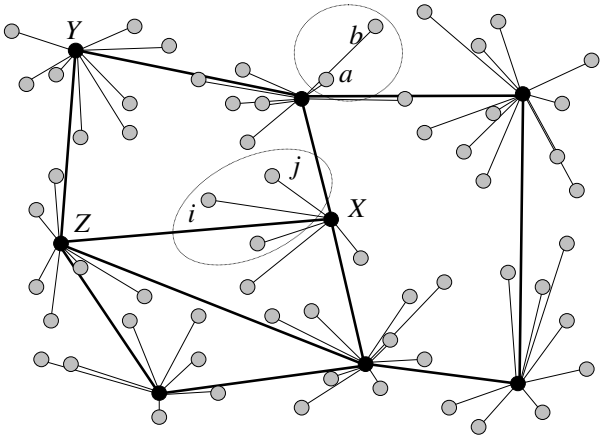

Fig. 3. A practical network design. Path length is limited and there is some redundancy/fault tolerance in the core network.

For a fully-connected core network, i.e. with a link between each switch pair *(X, Y)*, the cost of the link *(X, Y)* is given by

$$c_l(X,Y) = f_l(\sum_{i\in\Gamma_X}\sum_{j\in\Gamma_Y} t_{ij}, d_{XY}) \tag{4}$$

with an equivalent cost $c_l(Y,X)$ in reverse. Define $\Omega_{XY} = 1$ if there is a link between X and Y, $\Omega_{XY} = 0$ otherwise. (The node sets Γ_X, Γ_Y, …, and the connection matrix $\boldsymbol{\Omega} = (\Omega_{XY})$ fully describe any given solution.)

The total cost of the (fully-connected) network can then be calculated as

$$c^* = \sum_X \left[\begin{array}{l} \sum_{i\in\Gamma_X} (c_l(i,X) + c_l(X,i)) \\ + c_s(X) \\ + \sum_Y c_l(X,Y) \end{array} \right] \tag{5}$$

If the link from switch X to switch Y is not present ($\Omega_{XY} = 0$) this results in a saving of

$$f_l(\sum_{i\in\Gamma_X}\sum_{j\in\Gamma_Y} t_{ij}, d_{XY}) . \tag{6}$$

However its traffic must be redirected via switches Z_1, Z_2, … The cost of each affected switch, Z_1, Z_2, … will increase to

$$\begin{array}{l} f_s(\sum_{i\in\Gamma_{Z_1}}\sum_{j=1}^{n}(t_{ij}+t_{ji}) + \sum_{i\in\Gamma_X}\sum_{j\in\Gamma_Y} t_{ij}), \\ f_s(\sum_{i\in\Gamma_{Z_2}}\sum_{j=1}^{n}(t_{ij}+t_{ji}) + \sum_{i\in\Gamma_X}\sum_{j\in\Gamma_Y} t_{ij}),\ldots \end{array} \tag{7}$$

and link costs to

$$\begin{array}{l} f_l((\sum_{i\in X}\sum_{i\in Z_1} t_{ij} + \sum_{i\in X}\sum_{i\in Y} t_{ij}), d_{XZ_1}), \\ f_l((\sum_{i\in Z_1}\sum_{i\in Z_2} t_{ij} + \sum_{i\in X}\sum_{i\in Y} t_{ij}), d_{Z_1Z_2}), \\ \ldots \\ f_l((\sum_{i\in Z_r}\sum_{i\in Y} t_{ij} + \sum_{i\in X}\sum_{i\in Y} t_{ij}), d_{Z_rY}) \end{array} \tag{8}$$

for each capacity-enlarged link, (X, Z_1), (Z_1, Z_2), …, (Z_r, Y) where r is the degree of redirection for (X, Y) ($r=0 \Leftrightarrow \Omega_{XY}=1$). The calculation is repeated for each (X, Y) with $\Omega_{XY} = 0$. (If $\Omega_{XY} = 0$ implies $\Omega_{YX} = 0$ then the adjustments in (6, 7 & 8) are replicated in reverse but this is not assumed here.) The total network cost c^* can be recalculated accordingly. The removal of a link will result in an overall saving if appropriate spare capacity can be found on the switches and links through which its traffic is redirected (see Section 7).

(A distinction is made here between *redirection* and *rerouting*. Redirection is part of the topological design process by which required link capacities are estimated. Rerouting is a dynamic process taking place in real time on network switches. The use of redirection in design does not prohibit dynamic rerouting in operation.)

c^*, however, is a complex calculation, based on link costs that vary with network topology. Significantly, small changes to a topology (such as moving a node to a different parent switch) have consequential effects across the network and require a full re-evaluation of the total cost. Conventional local search techniques [4][5][6] work well when the effects of a local change can be calculated locally in terms of a change in cost (such as the insertion/removal of a link of fixed cost). Their complexity is increased if the cost function must be recalculated for each perturbation and their power diminishes rapidly

```
DD(n):
Make every node a switch                          // Initial solution
repeat
   Connect switches as a full-mesh
   Δs = 0
   Calculate c*                                   // Equation 5
   for each switch X do                           // Look to drop switches
      begin
         Calculate c*(X)                          // Equation 5
         repeat
            Δ1 = 0
            for each link (Y,Z) do
               begin                              // Look to drop links
                  Calculate Δ = c*(X)-c*(X,Y,Z)
                  if Δ > Δ1 then                  // Equations 6,7,8
                     Y* = Y, Z* = Z, Δ1 = Δ
               end
            if Δ1 > 0 then
               begin                              // Drop 'worst' link
                  Remove link (Y*,Z*)
                  c*(X) = c*(X,Y*,Z*)
               end
         until
            Δ1 = 0                                // No further link savings
         Δ = c* - c*(X)
         if Δ > Δs then
            X* = X, Δs = Δ
      end
   if Δs > 0 then
      Remove switch X*                            // Drop 'worst' switch
until
   Δs = 0                                         // No further switch savings
repeat                                            // Repeat link drops for final switch set
   Δ1 = 0
   for each link (Y,Z) do
      begin
         Calculate Δ = c*-c*(Y,Z)
            if Δ > Δ1 then
               Y* = Y, Z* = Z, Δ1 = Δ
      end
   if Δ1 > 0 then
      begin
         Remove link (Y*,Z*)
         c* = c*(Y*,Z*)
      end
until
   Δ1 = 0                                         // Final solution
```

Fig. 4. The Double-Drop (DD) network design process. $c^*(X)$ is the cost of the current solution with switch X removed. $c^*(Y,Z)$ is the cost of the current solution with the link (Y,Z) removed. $c^*(X,Y,Z)$ is the cost of the existing solution with X and (Y,Z) removed.

4. Local Search with Variable Costs

Theoretical search routines do not work well for the variable cost problem outlined here. There are n^{n-2} possible trees on n nodes [9] and a number of connected networks given recursively by

$$\Phi_n = 2^{n(n-1)/2} - \sum_{\mu=1}^{n-1} \frac{(n-1)!\Phi_\mu 2^{(n-\mu)(n-\mu-1)/2}}{(\mu-1)!(n-\mu)!} \qquad (9)$$

[10]. Both expressions are exponential, implying that exhaustive search is impractical for larger n.

An approach favoured by practical network designers, although its origins are uncertain, is outlined in Fig. 4. Starting from a full-switch/fully-connected network, the 'Double-Drop' (DD) algorithm of Fig. 4. tries candidate switches for removal from the current solution. With each trial switch removed, links are experimentally dropped in a similar manner. The algorithm is essentially 'greedy' but in a nested, local-search form. The network cost c^* and perturbated costs $c^*(X)$, $c^*(Y,Z)$ and $c^*(X,Y,Z)$ are calculated as in Section 3. There is an assumption that nodes are connected to their nearest switch.

```
RS(m):
min = MaxVal                                      // Some arbitrarily large value
for each node pair i, j (1≤i≠j≤m) do
   if dij < min then                              // Find closest pair
      i* = i, j* = j, min = dij
xk = (wi*xi* + wj*xj*) / (wi* + wj*)
yk = (wi*yi* + wj*yj*) / (wi* + wj*)
wk = wi* + wj*                                    // Replace by single node
for each node, η (η ≠ i*,j*) do
   begin                                          // with representative traffic,
      dkη = (wi*di*η + wj*dj*η)                   // coordinate and distance
            / (wi* + wj*)                         // characteristics
      dηk = (wi*dηi* + wj*dηj*) / (wi* + wj*)
   end
```

Fig. 5. A single reduction step, *RS(m)*. The closest modes ($min_{1\le i\ne j\le m} d_{ij}$) are replaced by a single node, representing the original pair in terms of traffic (weight), location and distance from the remaining nodes.

DD is a practical algorithm in that it deals with costs that vary with network topology. Its simple structure also minimises search iterations. Its major drawbacks are:

- It is unlikely to be particularly accurate since it removes switches and links in an entirely greedy manner with no consideration for a wider search neighbourhood,
- It is still computationally complex it its consideration of all combinations of node and link drops at each stage,
- Its complexity is increased further by the need to completely recalculate the cost function for each perturbation.

A natural extension to the DD process, to overcome the shortcomings of greedy search, is to introduce larger search neighbourhoods through (e.g.) tabu search and

simulated annealing. However, such refinements, whilst addressing the first problem, simply compound the second. For larger problems (n), DD has typically proved be the only viable search process.

5. A Geometric Reduction Approach

An alternative design method is proposed for large networks that eliminates a large number of iterations, branches and cost calculations. It uses the traffic values t_{ij} and distances d_{ij} to geometrically reduce the network in size. 'Conventional' optimisation then proceeds on the reduced version.

Define the *weight* of each node to be its total traffic load:

$$w_i = \sum_{j=1}^{n} (t_{ij} + t_{ji}) \qquad (10)$$

and note that this value is constant for any solution topology. We also require each node i to be defined by its Cartesian coordinates, (x_i, y_i). Then define a single *reduction step*, *RS(m)*, acting on m nodes, as in Fig. 5,

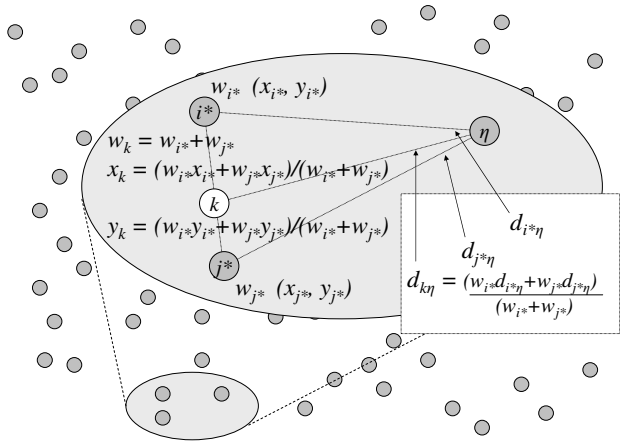

Fig. 6. A single reduction step, *RS(m)*. The two closest nodes ($i*$ and $j*$) are replaced by a single node (k), representing the location and traffic of the originals.

RS(m) finds the closest two nodes, as defined by distances d_{ij} and replaces them by a single, representative node, biased by the weights w_i and w_j. The original m nodes are replaced by a representative *m-1* in this single step. The procedure is shown graphically in Fig. 6. *RS(m)* is the essential component in a compound algorithm that can perform conventional optimisation on a network problem of reduced size. If *RS(m)* is repeated $n - q$ times, the original network problem of size n will be replaced by a representative one of size q, as shown in Fig. 7. These q nodes can be used in three ways to approximate an optimum solution – described in the next section. The complexity of the reduction process, a sequence of matrix searches, is bounded above by $O(n^3)$.

In principle, these replacement nodes could have been generated by a grid-based top-level division of the network into q regions then averaging x- and y-coordinates in each. However, the q nodes generated by bottom-up reduction described here are truly representative of the underlying problem. They will be distributed according to node clustering and traffic density, not uniformly with arbitrary region boundaries dividing natural node/traffic groups (see Appendix C). Another problem with this method is that fixed partitions produce a fixed number of representative nodes. The nodes produced by p partitions are independent of those produced by $p+1$, etc. In contrast, representative reduction generates each iteration from the previous one. An illustrative example is given at the end of the results section (8).

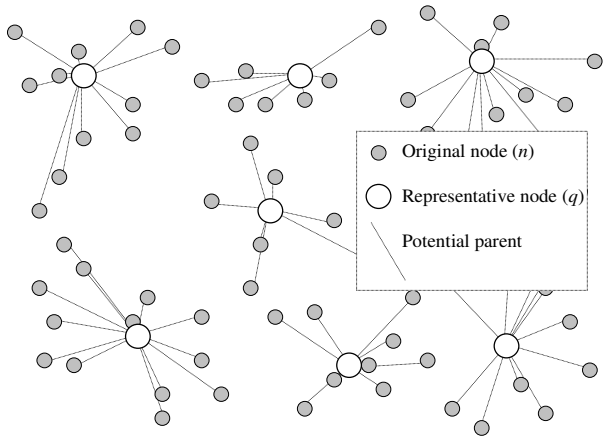

Fig. 7. A reduced network problem. The q nodes can be used immediately as approximations for switches or as a reduced node set on which to conduct exhaustive or DD search. Light lines show potential parents.

6. Optimisation on a Reduced Network Problem

Fig. 7 shows q replacement nodes at 'greenfield' sites, i.e. not true (original) node locations. Assuming the intention is to site switches at existing locations, define the step *Rel(q)* to be the process of relocating the q representative nodes to their nearest true node. A formal description of this simple process together with a demonstration that two representative nodes cannot relocate to the same true node is given in Appendix A. If greenfield sites *are* permitted

then the step may be omitted from the final process.

For any given computer upon which optimisation is to be performed (i.e. its processor power) we define the *optimisation limit values*. n_{ES} is the maximum number of nodes for which exhaustive search is feasible and n_{DD} the maximum number of nodes for which double-drop is feasible. Clearly, $n_{DD} > n_{ES}$ but actual values depend on the time available. For a given (time) limit, the value of n_{ES} may be derived empirically (Section 8) or calculated exactly from the known complexity of the exhaustive search process given in Appendix B. The double drop algorithm of Section 3, however, is indefinitely iterative: n_{DD} is best derived by experimentation.

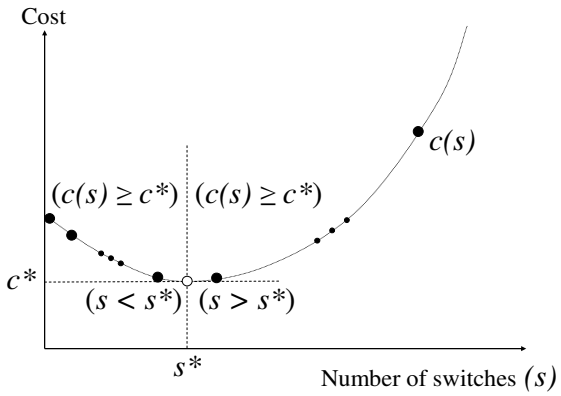

Fig. 8. Cost of optimum solution for fixed numbers of switches. There is a global minimum at $s = s^*$, the optimum number of switches.

Three compound heuristics are outlined in the subsections that follow and given formally in Fig. 9.

6.1. Reduction to Exhaustive Search (RES)

This is a simple, intuitive process. Reduce the number of nodes to n_{ES}, relocate to true positions and optimise to find switches and core network through exhaustive search. An exhaustive search algorithm is given in Appendix B.

6.2. Reduction to Double-Drop (RDD)

This is equally simple. Reduce the number of nodes to n_{DD}, relocate to true positions and perform double-drop optimisation to find switches and core network.

6.3. Reduction to Switch Location (RSL)

This is not so straightforward. The ideal is as follows. Reduce the number of nodes by one each time, immediately relocate to true positions (a single step only for the new node), explicitly make each node a switch and optimize on the core network only. Calculate cost (Equations 5, 6, 7 & 8). Repeat while cost decreases. The stopping principle assumes a concave cost function. (Let *c(s)* be the cost of the best network solution with *s* switches and let *s** be the optimum number of switches. Then $c^* = c(s^*)$ and for $s \neq s^*$, $c^* \leq c(s)$. The general form of *c(s)* is shown in Fig. 8.)

However, this would be an extremely complex approach. Evaluating each of the Φ_m core networks for each decreasing value of *m* switches (starting with $m = n$) is comparable with exhaustive search for complexity. To avoid this, we adopt the heuristic approach of only evaluating the cost of a fully-connected (mesh) core network. In Fig. 9, *COpt(m)* is the process of finding the *m* switches with the cheapest full-mesh core network and *co(m)* is the cost of this core network. The *optimal* core network is only calculated for the final switch set (*Opt(m)*).

7. Networks and Cost Functions

The algorithms introduced in this paper are compared in the next section. Two types of test instances were used: computer-generated and real-world. It is known [11] that certain algorithms can favour problem instances with parameters taken from particular statistical distributions so every attept is made to consider a variety of situations and characteristics.

```
RES:                RDD:                RSL:
  m = n               m = n               m = n
  repeat              repeat            repeat
    RS(m)               RS(m)             RS(m)
  until               until               Rel(m)
    m = n_ES            m = n_DD          COpt(m)
  Rel(m)              Rel(m)            until
  ES(m)               DD(m)               co(m) >
                                            co(m+1)
                                      m = m+1
                                      Opt(m)
```

Fig. 9. Three compound algorithms. RES: Reduce then apply exhaustive search; RDD: Reduce then apply double-drop; RSL: reduce and optimise directly on switches. For *RSL*, *co(m)* is the cost of a fully-connected core network on a given *m* switches. *COpt(m)* is the process (Eqn. 5) of finding the optimum set of *m* switches assuming a fully-connected core network and *Opt(m)* defines the process (Eqns. 6, 7 & 8) of finding (or approximating) the optimum core network, with cost *c(m)*, for a *given m* switches.

7.1. Computer-Generated Instances (CG)

Random generation of test instances is straightforward but must be appropriate and realistic. Just over *4,000* instances were produced with numbers of nodes (*n*) between *10* and *100,000*. Node positions were randomly taken from the *[0,1]* unit square but with reference to between *0* and *25 cluster points* (*cp*) and a *cluster coefficient* (*cc*) of between *0* and *1* (theoretically). A *cc* of

1 forces all nodes to be coincident on cluster points. A *cc* of *0* allows nodes to be placed anywhere – a uniform distribution across the unit square. *cp* and *cc* were randomized uniformly. Fig. 10 shows three examples.

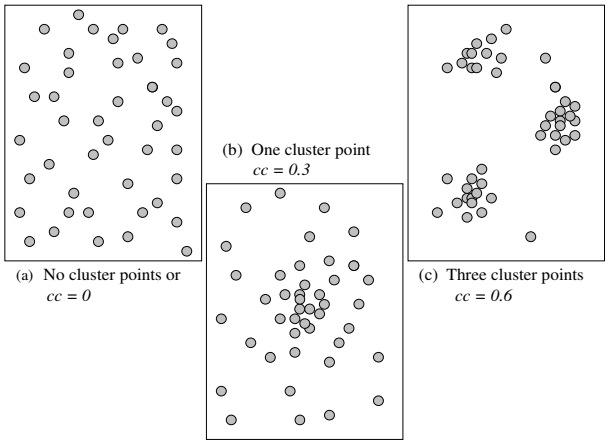

Fig. 10. Randomly generated node positions. (a). No clustering. (b). Moderate clustering around one point. (c). Heavy clustering about three points.

End-to-end traffic figures between each node pair were independently randomized on the interval *[0,1]* according to (both, separately) a uniform distribution (*U*) and a normal distribution (N_σ) with mean *0.5* and standard deviation (σ) between *0.005* and *0.25*.

Realistic link and switch costs are more complex – even for randomly generated instances. The benefits of redirecting traffic between switches, calculated in section III, are only positive if spare capacity can be found on existing links and switches to offset the additional cost of connection and switching. Real link and switch costs increase in discrete steps. General principles are given in [12] and formulated in [13]. Our cost functions are based on this approach and are described as follows. We initially consider transmission and switching sizes/costs together; they have similar characteristics at this level. We also use the general term *channel* to indicate a link/switch of a known size.

At low traffic levels, the smallest available channels will be used. Each will have a capacity of s_0. At some (traffic) limit, t_0, however, it will be preferable (cheaper) to use a single (larger) channel of size s_1. $s_1 > t_0$. For traffic, $0<t\leq t_0$, $\lfloor t/s_0 \rfloor+1$ channels will be required. ($\lfloor..\rfloor$ represents the integer part.) For traffic, $t_0<t\leq t_1$, $\lfloor t/s_1 \rfloor+1$ channels will be necessary, up to a limit of t_1, etc. This process is finite. There will be a largest channel of some size s_χ. Larger values of t will be carried by channels in multiples of s_χ. Fig. 11 shows $\chi=2$. In the case of transmission costs, the values of t_0, s_0, t_1, s_1, t_2, etc., will depend on link *Quality of Service* (*QoS*) and may be calculated or derived from tables. Switches may or may not have QoS parameters. In their absence, the values of t_0, s_0, t_1, s_1, t_2, etc. are fixed (for a given application).

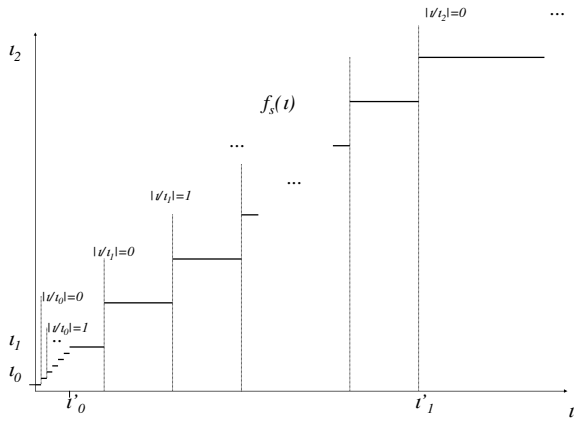

Fig. 11. Stepped cost function. Channels have fixed, maximum capacities and can be deployed in integer multiples of these values.

The cost of each channel of increasing capacity s_0, s_1, s_2, etc. will increase with t, but less than linearly. Also, transmission costs must take into account the length of the link. We may realistically model our cost functions, $f_s(T)$ and $f_l(t,d)$ as

$$f_s(T) = T^\xi \tag{11}$$

for switching and

$$f_l(t,d) = t^\xi d^\zeta \tag{12}$$

for transmission. $0<\xi<1$. In principle, $\zeta>0$. However, values of $\zeta>2$ are excessive in practice and we consequently restrict our test ranges to $0<\zeta<2$. Test instances with different values of t_0, s_0, t_1, s_1, t_2, … ξ and ζ are summarized in the results given in Section 8.

7.2. Real-World Instances (RW)

Four real network problems were also studied. Actual network data in the form of node locations, traffic requirements and link/switch costs were provided as follows:

Case 1: A Frame-Relay network of 78 nodes with estimated traffic flows, allowing the traffic matrix to be approximated.

Case 2: A Frame-Relay network of 103 nodes with known (measured) traffic flows, allowing the traffic matrix to be calculated.

Case 3: An ATM network of 221 nodes with unknown traffic flows. The traffic matrix is taken as being constant

for all node pairs.

Case 4: An IP network of 491 nodes with known (measured) traffic flows, allowing the traffic matrix to be calculated.

For reasons of commercial confidentially, it is not possible to release precise details of locations, traffic and costings. It is, however, acceptable to summarise results as in the next section.

8. Comparing Compound Algorithms

Five algorithms are considered and their complexity and accuracy compared.

ES: Exhaustive Search (Appendix B)

DD: Double-Drop (Section 3)

RES: Reduction to Exhaustive Search (Section 6)

RDD: Reduction to Double-Drop (Section 6)

RSL: Reduction to Switch Location (Section 6).

Table 1. Run-Times for Exhaustive Search (ES) and Double Drop (DD)

n	*ES* run-time (*s*)	*DD* run-time (*s*)
6 (CG)	< 1	< 1
8 (CG)	< 1	< 1
10 (CG)	< 1	< 1
11 (CG)	3.8	< 1
12 (CG)	77,123	< 1
13 (CG)	3×10^8 *	< 1
15 (CG)	4×10^{16} *	< 1
20 (CG)	1×10^{42} *	< 1
25 (CG)	-	< 1
30 (CG)	-	< 1
40 (CG)	-	< 1
50 (CG)	-	7.2
55 (CG)	-	242
60 (CG)	-	80,880
65 (CG)	-	269,385
70 (CG)	-	-

Run times in seconds. Actual timings except * calculated. CG: Computer Generated instances. RW: Real-World cases.

At the end of this section, we also consider *PES*, *PDD* and *PSL*, the fixed partition equivalents (Appendix C) of RES, RDD and RSL. However, these prove to be poorer performers in each case and are not considered in detail.

We deal first with run times. All are based on each algorithm, coded in C++, running on dual 3.4GHz Intel Xeon 64 bit processors with 8GB SDRAM. Generated instances (CG) are averaged, test cases (RW) given explicitly. Distances d_{ij} were calculated once then stored for efficiency. Table 1 compares run times for ES and DD.

Table 1 shows that, taking *one day* (*84,400s*) as the acceptable limit for program run time, the optimisation limit values are $n_{ES} = 12$ and $n_{DD} = 60$. Other values may be used but we stick with these in this paper. We use the values in subsequent tests to perform full or partial optimisation at the earliest opportunity – i.e. on the largest possible number of nodes. There was little variance, and no apparent correlation, in the run times for different values of *cp*, *cc* and σ (or t_0, s_0, t_1, s_1, t_2, … ξ and ζ) (Section 7). Table 2 gives the run times of RES, RDD and RSL using these values of n_{ES} and n_{DD}.

Table 2. Run-Times for Reduction to Exhaustive Search (RES), Reduction to Double Drop (RDD) and Reduction to Switch Location (RSL)

n	*RES* run-time (*s*)	R*DD* run-time (*s*)	RSL run-time (s)
1,000 (CG)	77,123	80,880	< 1
5,000 (CG)	77,123	80,880	< 1
10,000 (CG)	77,123	80,880	< 1
25,000 (CG)	77,124	80,880	1
50,000 (CG)	77,129	80,882	24
100,000 (CG)	78,502	81,442	1,510

Run times in seconds. $n_{ES} = 12$ and $n_{DD} = 60$. CG: Computer Generated instances.

The dominance of the ES and DD components in RES and RDD is clear. RSL, with its simplified core network heuristic, is considerably less complex for all but the very largest values of *n*. Once again, there was no noticeable difference for different values of *cp*, *cc* and σ (or t_0, s_0, t_1, s_1, t_2, … ξ and ζ).

In measuring the accuracy of the different optimisation methods, there is a large quantity of data/results to report and it is necessary to summarise for brevity. Although, among the large numbers of tests, there was some fluctuation for different values of cost parameters t_0, s_0, t_1, s_1, t_2, … ξ and ζ (Section 7), there is no clear pattern and the variance is attributed to statistical uncertainty.

Table 3. Performance of Reduction to Switch Location (RSL) relative to Exhaustive Search (ES) for small problems

n	*c**	✔	*n*	*c**	✔
6 (CG)	0.03	94.2	10 (CG)	0.16	94.0
7 (CG)	0.05	94.1	11 (CG)	0.20	93.8
8 (CG)	0.11	94.1	12 (CG)	0.23	93.8
9 (CG)	0.14	94.0	13 (CG)	0.25	93.7

Percentage (%) cost (*c**) increase of RSL over ES and percentage (%) of RSL finding same (optimal) solution as ES (✔). CG: Computer Generated instances.

There was, however, some clear relationship between the size of the problems in question (*n*), the number of cluster points (*cp*), the cluster coefficient (*cc*) and the distribution

from which the end-to-end traffic matrix was taken (uniform or normal with distribution σ) and the relative performance of the different algorithms and these are summarized in what follows.

Table 4. Performance of RES with RSL for Different Traffic Characteristics

n	*U*	*N(0.25)*	*N(0.1)*	*N(0.005)*
10 (CG)	1.8	1.8	1.7	1.7
30 (CG)	2.3	2.2	2.0	2.0
100 (CG)	2.7	2.5	2.4	2.2
300 (CG)	4.0	3.9	3.7	3.6
1,000 (CG)	5.2	4.9	4.7	4.6
3,000 (CG)	7.0	6.6	6.5	6.2
10.000 (CG)	8.2	7.8	7.4	7.1
30,000 (CG)	9.3	9.0	8.5	8.2
100,000 (CG)	11.4	10.5	10.2	9.9

Percentage (%) cost (c^*) saving of RSL over RES for traffic values drawn from uniform and normal distributions. $cp = cc = 0$. CG: Computer Generated instances.

We begin with a set of small tests, comparing ES with RSL. ES can only run up to $n = 12$ and RSL is the only one of the three reductive algorithms (RES, RDD and RSL) that is faster than ES for problems of this size.

Table 5. Performance of RES and RDD with RSL for Cluster Characteristics $cp = 0$, $cc = 0$

n	*RSL*	*RDD*
10 (CG)	1.8	2.4
30 (CG)	2.3	3.2
100 (CG)	2.7	5.1
300 (CG)	4.0	8.2
1,000 (CG)	5.2	10.2
3,000 (CG)	7.0	12.8
10.000 (CG)	8.2	15.0
30,000 (CG)	9.3	17.2
100,000 (CG)	11.4	21.5

Percentage (%) cost (c^*) saving of RSL and RDD over RES for $cp = cc = 0$. CG: Computer Generated instances.

Table 3 shows that the inaccuracies of RSL, relative to ES, are small at this level but increase with n. It is conjectured, but cannot be tested, that these inaccuracies will continue to increase with larger n. For these small problems, there was no apparent significance to the parameters cp, cc and σ.

We turn now to traffic variance. For our tests, the traffic requirement between end nodes i and j, t_{ij}, is taken from two distributions:

- U: t_{ij} drawn randomly, uniformly from the interval $[0,1]$, independently for each (i,j), and
- N_σ: t_{ij} drawn randomly on the interval $[0,1]$ from a normal distribution with mean 0.5 and variance σ^2, independently for each (i,j)

U gives very random traffic. N_σ, for small σ, gives extremely self-similar traffic. Table 4 compares the behaviour of the RES and RSL reductive algorithms for different traffic characteristics with no clustering.

Table 6. Performance of RES and RDD with RSL for Cluster Characteristics $cp = 5$, $cc = 0.5$

n	*RSL*	*RDD*
10 (CG)	1.9	2.4
30 (CG)	2.4	3.3
100 (CG)	2.9	5.0
300 (CG)	4.3	8.2
1,000 (CG)	5.5	10.0
3,000 (CG)	7.4	12.9
10.000 (CG)	8.8	14.7
30,000 (CG)	9.7	16.8
100,000 (CG)	11.9	21.7

Percentage (%) cost (c^*) saving of RSL and RDD over RES for $cp = 5$, $cc = 0.5$. CG: Computer Generated instances.

Table 4 suggests that the cruder reductive approach of RES is less poor for problems with heavily self-similar traffic, due to the likelihood that a larger number of solutions have relatively close costs. When traffic is diverse, there is a greater tendency for RES to miss the optimal solution by a greater margin. In general RSL outperforms RES by a greater margin anyway for larger problems. Table 4 is a summary and the difference is slight. These patterns were observed in the problems with clustering features that follow but are omitted for brevity (uniform traffic distributions are assumed).

Table 7. Performance of RES and RDD with RSL for Cluster Characteristics $cp = 15$, $cc = 0.4$ & $cp = 15$, $cc = 0.8$

n	*cc=0.4*	*RSL*	*RDD*	*cc=0.8*	*RSL*	*RDD*
10 (CG)		2.0	2.2		2.1	2.1
30 (CG)		2.5	3.1		2.7	3.0
100 (CG)		3.1	4.6		3.3	4.4
300 (CG)		4.8	7.7		4.9	7.6
1,000 (CG)		5.8	9.5		6.2	9.3
3,000 (CG)		7.9	12.2		8.2	11.9
10.000 (CG)		9.3	14.3		9.7	13.9
30,000 (CG)		10.0	16.3		10.2	15.9
100,000 (CG)		12.5	21.3		12.8	20.6

Percentage (%) cost (c^*) saving of RSL and RDD over RES for $cp = 15$, $cc = 0.4$ & $cp = 15$, $cc = 0.8$. CG: Computer Generated instances.

We now compare RSL, RES and RDD directly. In particular, we consider problems with different cluster characteristics, cp and cc. Tables 5, 6, 7 and 8 compare RSL and RDD with RES for cp:cc pairs, 0:0, 5:0.5,

15:0.4, *15:0.8*, *25:0.4* and *25:0.8*. Although RDD consistently outperforms RSL, which in turn outperforms RES, it can be seen that, as the level of clustering increases, RSL performs comparatively better and RDD comparatively worse. Fig. 12 summarises Tables 5 to 8. Table 9 gives the equivalent results for the real-world cases, which broadly correlate with the computer-generated instances.

Table 8. Performance of RES and RDD with RSL for Cluster Characteristics *cp = 25, cc = 0.4* & *cp = 25, cc = 0.8*

n	*cc=0.4*	*RSL*	*RDD*	*cc=0.8*	*RSL*	*RDD*
10 (CG)		2.2	1.9		2.4	1.7
30 (CG)		2.7	2.8		2.8	2.6
100 (CG)		3.4	4.1		3.6	3.7
300 (CG)		5.2	7.2		5.4	6.9
1,000 (CG)		6.3	8.9		6.5	8.5
3,000 (CG)		8.3	11.8		8.4	11.6
10.000 (CG)		9.7	13.3		10.1	13.0
30,000 (CG)		10.6	15.2		10.9	14.7
100,000 (CG)		12.9	19.8		13.5	19.3

Percentage (%) cost (*c**) saving of RSL and RDD over RES for *cp =25, cc = 0.4* & *cp = 25, cc = 0.8*. CG: Computer Generated instances.

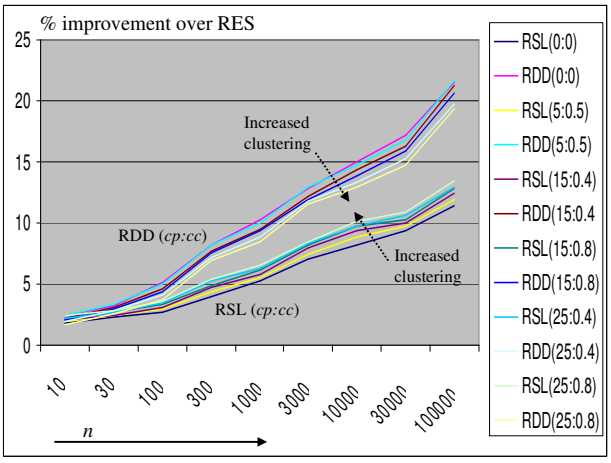

Fig. 12. Percentage improvement of RSL and RDD over RES for different cluster values of (*cp:cc*). RDD is always better than RSL, the difference increasing with *n*. However, RSL performs comparatively better with increased clustering and RDD comparatively worse.

Table 9. Performance of RES and RDD with RSL for Real-World Cases

n	*RSL*	*RDD*
78 (RW)	3.0	3.4
103 (RW)	3.1	6.2
221 (RW)	3.6	6.9
491 (RW)	5.1	8.9

Percentage (%) cost (*c**) saving of RSL and RDD over RES for Real-World (RW) cases.

Table 10. Performance of PES, PSL and PDD with RES, RSL and RDD

n	*PES/RES*	*PSL/RSL*	*PDD/RDD*
10 (CG)	0.3	1.2	0.7
30 (CG)	2.9	4.6	3.1
78 (RW)	6.2	8.9	5.3
100 (CG)	6.1	10.5	7.0
103 (RW)	5.6	9.1	8.4
221 (RW)	9.2	12.0	11.0
300 (CG)	8.8	18.0	10.3
491 (RW)	8.9	15.5	9.7
1,000 (CG)	11.7	22.1	13.3
3,000 (CG)	15.0	28.6	17.5
10.000 (CG)	18.2	35.5	20.6
30,000 (CG)	22.0	43.8	26.6
100,000 (CG)	27.8	55.1	33.2

Percentage (%) cost (*c**) increase of PES, PSL and PDD over RES, RSL and RDD. CW: Computer Generated instances (*cp = 0; cc = 0*). RW: Real-World cases.

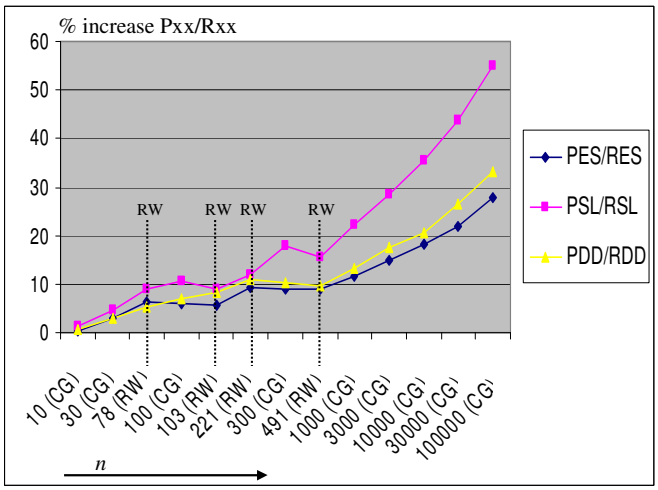

Fig. 13. Comparing PES, PSL and PDD with RES, RSL and RDD. RW cases do not behave exactly as CG instances but the relative performances remain clear.

Finally, we compare representative reduction with arbitrary partitioning. Table 10 shows the relative performance (percentage cost increase) of each of the compound partitioning algorithms, PES, PSL and PDD (Appendix C) with its reductive equivalent, RES, RSL and RDD. Both real and generated cases (with no clustering) are given. Representative reduction outperforms arbitrary partitioning in all cases (although there is an imperfect correlation between the CG and RW examples). Fig. 13 shows the trend graphically. PSL is particularly poor.

9. Conclusions

Theoretical, fixed-cost models of the network design process are simplistic. In the practical design of a real

network, both link and switch costs have to be considered and these costs are (at least partially) a function of (required) capacity. As this capacity depends upon the topology of the solution network, costs cannot be considered fixed and entered as input to a standard algorithmic solution. The further, implied difficulty that the cost function is not locally stable, and must be re-evaluated fully for each solution variant, increases the complexity considerably, particularly for large problems. Conventional construction or local search variants fail for one or both of these reasons.

Noting these objections, this paper initially considers two practical optimisation algorithms: exhaustive search (ES) and a doubly-iterative drop (DD). However, both have limits (n_{ES} and n_{DD}) on network size so additional techniques are needed to reduce larger problems to within their range. The top-down and bottom-up approaches are considered: firstly, partitioning (P) the plane into arbitrary areas and secondly, reducing (R) the problem representatively. In isolation, R is more complex than P but both are insignificant if ES and DD are running at their limits. In fact, P solutions are generally poorer.

R solutions have three variants: reduce down to exhaustive search (RES), reduce down to double-drop (RDD) and reduce directly down to switch location (RSL). There is an additional heuristic simplification involved in RSL (Section 6). The equivalent PES, PDD and PSL do not give good results (comparatively).

RES does not perform well, mainly due to a necessarily very small n_{ES}. RSL gives better results: its core network heuristic makes it the fastest of the three R approaches at the expense of some accuracy. RDD is the most accurate: $n_{DD} > n_{ES}$ outweighing the heuristic limitation of DD. If time permits, RDD would be the preferred method of solution for a large-scale network design problem. If less accurate results are required much faster (for example, if frequent re-optimisation is to be performed) then RSL is an acceptable compromise.

Appendix A: . Node relocation

The relocation process, *Rel(m)* or *Rel(q)*, relocates each of the reduced m or q representative nodes to its nearest original node. The algorithm for (the *Rel(m)* version of) this simple process is given in Fig 14.

The repeated application of *RS(m)* will place each representative node at the *weighted centre* of the original node set it represents (Fig. 15). The implications of this are that, taking any direction from representative node m (e.g. to another representative node), there will be at least one original node, in m's node set, in that direction that is closer to m than any node not in m's node set. Each representative node thus relocates to an original node in its own set and all representative nodes locate to different original nodes. An extreme case is where no reduction has taken place for a given 'replacement' node, in which case it 'relocates' to the single original node with which it is still coincident.

```
Rel(m):
for each replacement node k (1≤k≤m) do
   begin
      j = 1, min = d1k
      for each original node i (2≤i≤n) do
         if dik < min then
            j = i, min = dik
      relocate k to j
   end
```

Fig. 14. The relocation process, *Rel(m)*, relocating each of the m representative nodes to its nearest original node.

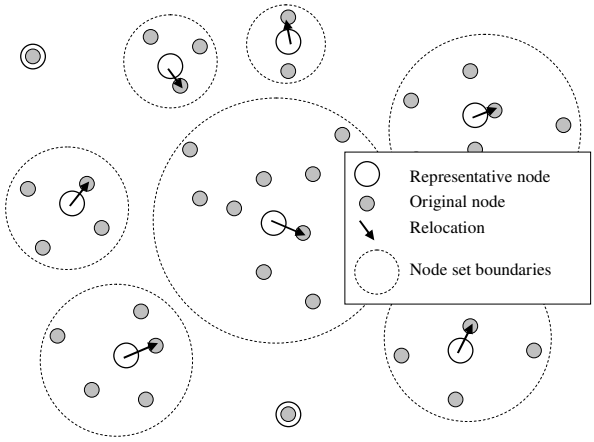

Fig. 15. The relocation process, *Rel(m)* on m representative nodes. No two representative nodes relocate to the same original node.

Appendix B: . Exhaustive Search

An exhaustive search algorithm, *ES(n)* or *ES(m)*, on n original or m replacement nodes tries every combination of switches and core networks and returns the configuration of least cost. *ES(n)* is given in Fig. 16. The use of equations 5, 6, 7 & 8 is implicit.

It is assumed in *ES(n)* as given that nodes connect to their nearest switch. In the majority of, but not all cases, this will yield the optimum solution. In isolated cases, however, it may be that connecting certain nodes to

switches other than the closest may better use capacity on the distant switch, avoid an extra step cost on the nearer, and justify the slight extra cost of the longer link. *ES(n)* (or *ES(m)*) are easily adapted to include this extra search level but their complexity increases substantially.

```
ES(n):
min = MaxVal                          // Some arbitrarily large value
for each m (1≤m≤n) do
   for each node subset S = {s:1≤m≤n} (|S|=m) do
      for each connected core network
         Ξ = {(i,j):1≤i,j≤n and i,j∈S} do
            if c* < min then
               min = c*
               S* = S
               Ξ* = Ξ
```

Fig. 16. The exhaustive search process, *ES(n)*. All combinations of switches and core network are considered. The process returns the optimal switch set, S^*, and the optimal core network, Ξ^*.

There are ${}^{n}C_{m} = \frac{n!}{m!(n-m)!}$ choices of m switches from n nodes and, adapting the expression from section 4, $\Phi_m = 2^{m(m-1)/2} - \sum_{\mu=1}^{m-1} \frac{(m-1)!\Phi_\mu 2^{(m-\mu)(m-\mu-1)/2}}{(\mu-1)!(m-\mu)!}$ connected core networks for each switch set. There are m^{n-m} ways of connecting n-m non-switch nodes to m switches so the total number of network configurations to be considered by *ES(n)* is $\sum_{m=1}^{n} {}^{n}C_{m}\Phi_{m}m^{n-m}$, which simplifies to $\sum_{m=1}^{n} {}^{n}C_{m}\Phi_{m}$ if nodes are constrained to connect to their nearest switches. A similar expression applies to *ES(m)*. This allows the value of n_{ES} in Section 6 to be calculated. However, both are doubly exponential in complexity.

Appendix C: . Top-down versus Bottom-up Clustering

The reduction process described in this paper builds a set of representative nodes by bottom-up iterative replacement. In so doing, it produces the best possible reduced representation of the original nodes in terms of both traffic density and position and can take into account either Euclidean or non-Euclidean distances.

There is of course another, simpler way of generating these replacement nodes, which has been used for various geometric optimisation problems over the years (e.g. [14]). The Partitioning method, *P(m)*, divides the plane up into m regions, $R_1, R_2, \ldots, R_m$, and calculates the co-ordinates of the representative node for each region R_i as

$$(\sum_{j\in R_i} w_j x_j / \sum_{j\in R_i} w_j, \sum_{j\in R_i} w_j y_j / \sum_{j\in R_i} w_j) . \qquad (13)$$

```
PES:                PDD:                PSL:
m = n               m = n               m = n
repeat              repeat              repeat
  P(m)                P(m)                P(m)
until               until                 Rel(m)
  m = n_ES            m = n_DD              COpt(m)
Rel(m)              Rel(m)              until
    ES     (m)              DD(m)          co(m) >
                                             co(m+1)
                                        m = m+1
                                        Opt(m)
```

Fig. 17. The three partitioned equivalents of the RES, RDD and RSL algorithms. PES: Partition then apply exhaustive search; PDD: Partition then apply double-drop; PSL: (Partition to Switch Location). Reduce and optimise directly on switches.

On this basis, we can define the partitioned equivalents of RES, RDD and RSL as in Fig. 17.

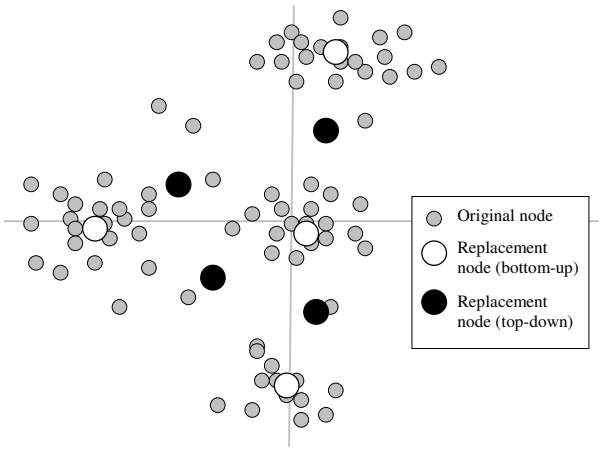

Fig. 18. Comparing the bottom-up and top-down node replacement policies. Arbitrary divisions of the plane do not give good representative nodes and cannot take non-Euclidean distances truly into account.

Figure 18, however, shows typical results for this method compared with representative reduction. Not only are the partitioned replacement nodes poor representatives of the original clusters, equation (13) does not take into account (particularly non-Euclidean) distances in the manner of *RS(m)* (Fig. 5). For our purposes, reduction proves to be stronger than partitioning at no practical increase in complexity.

References

[1] R.C. Prim, "Shortest Connection Networks and some Generalizations," *Bell Systems Tech. J.* Vol. 36, pp1389-1401, 1957.

[2] M.R. Garey and D.S. Johnson, *Computers and Intractability: A guide to the theory of NP-completeness*, W.H. Freeman, New York, 1979.

[3] A. Kershenbaum and W. Chou, "A Unified Algorithm for Designing Multidrop Teleprocessing Networks", *IEEE Trans. Communications*, Vol. COM-22, No. 11, pp1762-1772, 1974.

[4] F. Glover and M. Laguna, "Tabu Search," in *Modern Heuristic Techniques for Combinatorial Problems*, Reeves, Ed. New York, Wiley, 1993, pp70-150.

[5] E.H.L. Aarts and J.K. Lenstra, *Local Search in Combinatorial Optimization*, Wiley, 1997.

[6] G. Winter, J. Periaux and M. Galan (Eds.), *Genetic Algorithms in Engineering and Computer Science*, Wiley, 1995.

[7] M. Dorigo, G. Di Caro and L.M. Gambardella, "Ant Algorithms for Discrete Optimization," in *Artificial Life*, MIT Press, 1999.

[8] B.S. Verkhovsky, and Y.S. Polyakov, "Algorithms for Optimal Switch Location: Concave Cost Functions," in *Advances in Decision Technology and Intelligent Information Systems, Vol. IV*, Lasker, Ed. Int. Inst for Advanced Studies in Systems Research and Cybernetics, pp16-20, 2003.

[9] J.W. Moon, "Counting Labelled Trees", *Canadian Mathematics Congress*, Montreal, 1970.

[10] V. Grout, "Principles of Cost Minimisation in Wireless Networks", *Jnl. Heuristics*, Vol. 11, Issue 2, March 2005.

[11] V. Grout, "Initial Results from a Study of Probability Curves for Shortest Arcs in Optimal ATSP Tours with Application to Heuristic Performance", *Proc. 20th British Combinatorial Conference* (*BCC 2005*), University of Durham, UK, 10th-15th July 2005.

[12] R. Chan, *Wide Area Network Design: Concepts & Tools for Optimization*, Morgan Kaufmann, 1998.

[13] V. Gabrel, A. Knippel and M. Minoux, "Exact solution of multicommodity network optimization problems with general step cost functions," *Operations Research Letters*, Vol. 25, pp15-23, 1999.

[14] G. Gutin and A.P. Punnen, *"The Traveling Salesman Problem and its Variations"*, Kluwer, 2002.

Vic Grout was awarded the BSc(Hons) degree in Mathematics and Computing from the University of Exeter (UK) in 1984 and the PhD degree in Communication Engineering from Plymouth Polytechnic (UK) in 1988.

He has worked in senior positions in both academia and industry for twenty years and has published and presented over 120 research papers. He is currently a Reader in Computer Science at the University of Wales NEWI, Wrexham in the UK, where he leads the Centre for Applied Internet Research (CAIR). His research interests and those of his research students span several areas of computational mathematics, particularly the application of heuristic principles to large-scale problems in network design and management.

Dr. Grout is a Chartered Engineer, Scientist and Mathematician and a Fellow of the British Computer Society (BCS) and Institute of Engineering and Technology (IET(. He chairs the biennial international conference series on Internet Technologies and Applications (ITA 05 & ITA 07).

Stuart Cunningham was awarded his BSc degree in Computer Networks in 2001, and in 2003 was awarded the MSc Multimedia Communications degree with Distinction, both from the University of Paisley (UK). He is a Member of the British Computer Society and the Institute of Engineering & Technology. Stuart is also a member of the MPEG Music Notation Standards working group. Since 2003, he has been working at the University of Wales as a lecturer where he teaches audio visual computing and computer systems architecture. Stuart is also a PhD student at the University of Wales, studying under the supervision of Dr. Vic Grout.

Rich Picking has a BSc(Hons) degree in Computing and Operational Research from Leeds Polytechnic (UK, 1986), an MSc in Control Engineering and Information Technology (University of Sheffield, UK, 1987) and a PhD in Interactive Multimedia Interface Design from Loughborough University (UK) in 1996.

Dr. Picking has authored and edited numerous papers, proceedings and guides in various fields of Internet Technologies and Applications. He currently a Principal Lecturer in Computing in the School of Computing and Communications Technology within the Faculty of Business, Science and Technology at the University of Wales NEWI, Wrexham in the UK.