

# SPANNING TREE OBJECTIVE FUNCTIONS AND ALGORITHMS FOR WIRELESS NETWORKS

Mike Morgan and Vic Grout  
Centre for Applied Internet Research (CAIR)  
University of Wales, NEWI, Plas Coch Campus  
Wrexham, North Wales, LL11 2AW, UK  
{mi.morgan|v.grout}@newi.ac.uk

**Abstract** This paper considers various forms of objective function that may be applied in the calculation of spanning trees in different network situations. Conventional link and path cost approaches are compared to those based on switch or bridge costs more appropriate for wireless applications. Variant objectives are formulated and compared. Although efficient exact algorithmic approaches exist only for the link cost objectives, reasonable approximations for the switch/bridge equivalents are to be found with simple greedy heuristics and better results still through various forms of iterated local search such as tabu search and simulated annealing.

**Index Terms**— Wireless network optimization, Spanning trees, Objective functions, Algorithms, Heuristics

## I. INTRODUCTION: SPANNING TREE OBJECTIVE FUNCTIONS

*Spanning Tree Protocols* are integral in the efficient operation of many forms of multipart/area network [1]. They provide loop-free routing/delivery by establishing a single path between all node pairs. Finding a spanning tree (ST) of minimal ‘cost’ is clearly beneficial; however the relevant ‘cost’ to be minimized may vary between applications. In particular, in wireless applications, any approach based on link cost is clearly inappropriate.

Begin with a graph  $G = (V, E)$ ,  $|V|=n$ . The conventional *Minimum Spanning Tree (MST)* approach assigns a cost to each feasible link  $(i, j) \in E$  then seeks to minimize the sum of all link costs in the ST, that is to find the minimizing tree  $T^*$  such that

$$f_{MST}(T^*) = \sum_{i=1}^{n-1} \sum_{j=i+1}^n c_{ij} x_{ij}^{T^*} = \min_T f_{MST}(T) = \min_T \sum_{i=1}^{n-1} \sum_{j=i+1}^n c_{ij} x_{ij}^T \quad (1)$$

for all trees  $T$ , where  $c_{ij}$  is the cost of the link  $(i, j)$  and  $x_{ij}^T = 1$

if  $(i, j) \in T$  and  $x_{ij}^T = 0$  if  $(i, j) \notin T$ . The MST problem can be solved efficiently [2].

However, minimizing total link cost is rarely appropriate in a real-world environment. An alternative is to minimize path length - either maximum or average. If  $p_{ij}^T$  is the length of the path from  $i$  to  $j$  in  $T$  (the sum of the link costs over the path) then the *Minimum Maximum Path (MMP)* or *Minimum Average Path (MAP)* objectives are to find  $T^*$  such that

$$f_{MMP}(T^*) = \max_{ij} p_{ij}^{T^*} = \min_T f_{MMP}(T) = \min_T \max_{ij} p_{ij}^T \quad (2)$$

and

$$f_{MAP}(T^*) = \frac{1}{n(n-1)} \sum_{i=1}^{n-1} \sum_{j=i+1}^n p_{ij}^{T^*} = \min_T f_{MAP}(T) = \frac{1}{n(n-1)} \min_T \sum_{i=1}^{n-1} \sum_{j=i+1}^n p_{ij}^T \quad (3)$$

Again, efficient techniques exist for minimizing paths [3].

In these formulations and those that follow, symmetric network parameters ( $x_{ij}^T = x_{ji}^T$  and consequently  $p_{ij}^T = p_{ji}^T$  for all  $i, j$ ) are assumed. This is only for brevity: asymmetric values are equally valid.

## II. SPANNING TREE OBJECTIVES FOR WIRELESS NETWORKS

For wireless networks, link costs are best avoided entirely. Path lengths may still be minimized as in (2) and (3) but should ideally be based on switch/bridge cost, not link cost.

Define  $c_i$  to be the cost of the bridge  $i$  and  $z_i^T$  as:

$$z_i^T = \begin{cases} 1: \sum_{j=1}^n x_{ij}^T > 1 \\ 0: \sum_{j=1}^n x_{ij}^T = 1 \end{cases} \quad (4)$$

$z_i^T$  defines the active bridges in  $T$ .  $z_i^T=0$  identifies non-participatory ‘leaf’ nodes. A more realistic objective function for a wireless environment is then to find  $T^*$  such that

$$f_{MAB}(T^*) = \sum_{i=1}^n c_i z_i^{T^*} = \min_T f_{MAB}(T) = \min_T \sum_{i=1}^n c_i z_i^T \quad (5)$$

the *Minimum Active Bridge (MAB)* objective, or to redefine (2) and/or (3) in terms of switch/bridge costs, giving the *Minimum Maximum Bridge Path (MMBP)* and *Minimum Average Bridge Path (MABP)* objectives.

ADD

```
// Initialization
for all  $i \in V$  do  $s_i^N \leftarrow 0$ 
for all  $i, j \in V$  do  $x_{ij}^N \leftarrow 0$ 
find  $i$  such that  $v_i = \max_j v_j$ 
 $s_i^N \leftarrow 1$ 
// Growth
while there exists  $j$  such that  $s_j^N = 0$  do
{
  for all  $j \in V$  such that
     $b_j = 1$  and  $e_{ij} = 1$  and  $s_j^N = 0$  do
    {
       $x_{ij}^N \leftarrow 1$ 
       $s_j^N \leftarrow 1$ 
    }
  find  $i$  such that
     $v_i - \delta_i^N = \max_j (v_j - \delta_j^N)$ 
    where  $s_j^N = 1$ 
}
```

Figure 1. The Add algorithm for approximating the minimum-bridge solution

The  $f_{MST}$ ,  $f_{MMP}$ ,  $f_{MAP}$ ,  $f_{MMBP}$ ,  $f_{MABP}$  and  $f_{MAB}$  objectives are based on different underlying graph problems [4] and have different constrained complexities [5]. A simple greedy algorithm works well for the MST [2]. In principle, shortest path algorithms [3] deal with the MMP, MAP, MMBP and MABP. However, it is known from various routing studies [6] that individually optimized paths do not provide optimal solutions for the network as a whole and that the compound problem is more complex [4].

It is shown in this paper that minimizing  $f_{MAB}$  also minimizes or reduces  $f_{MMP}$ ,  $f_{MAP}$ ,  $f_{MMBP}$  and  $f_{MABP}$ . Simple but effective greedy ‘add’ and ‘drop’ algorithms for the MAB problem are given in [5]. This paper discusses improved iterated local search (‘meta-heuristic’) approaches to the MAB (and consequently MMP, MAP, MMBP and MABP) problem based on Tabu-Search (TS) and Simulated Annealing (SA) [7] principles. TS techniques are shown to give improved results for the MAB, MMP, MAP, MMBP and MABP problems. A

developmental SA approach is also considered.

The extension of these centralized algorithms to their distributed equivalents is considered in conclusion.

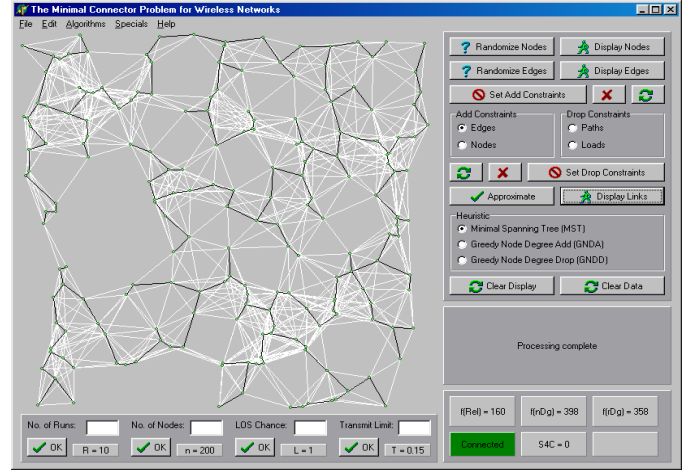


Figure 2. MST Spanning tree solution

### III. SPANNING TREE ALGORITHMS FOR WIRELESS NETWORKS

[5] gives a complete formulation of the problem of minimizing bridge (or *relay*) costs in a wireless network, including three separate objective functions for:

- minimizing the number of bridges,
- minimizing the degree of the active bridges,
- minimizing the degree of all network nodes.

For STs, as opposed to networks with redundant paths, these objectives are identical. [5] also details two classes of constraint that may be applied:

- *Add* constraints – identifying feasible edges and nodes (a feasible node is one that may act as a bridge),
- *Drop* constraints – identifying levels of redundancy and load restrictions.

Only add constraints are relevant for STs.

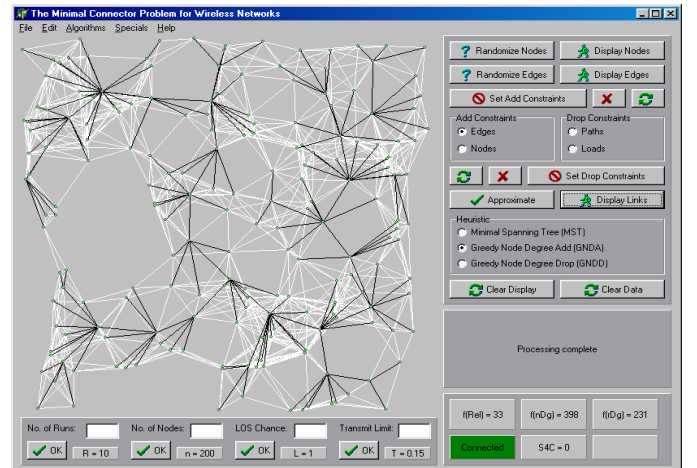


Figure 3. MAB Spanning tree solution

Finally, [5] offers two algorithms (*ADD* and *DROP*) to attempt optimum solutions. The ADD algorithm (Figure 1)

approximates the minimum bridge ST we require, producing significantly smaller  $f_{MAB}(T^*)$  values than an MST approach (Figures 2 and 3). It constructs a network,  $N$ , from an empty link set, using the *temporary spanning vector*,  $\underline{s}^N = (s_i^N : i \in V)$ , where  $s_i^N = 0$  initially for all  $i \in V$  and  $s_i^N = 1$  as  $i$  is included.  $v_i$  is the valency (degree) of the node  $i$  in the underlying graph  $G$  and  $\delta_j^N$  the current degree of  $j$  in  $N$ .  $b_j = 1$  if  $j$  is a viable bridge and  $e_{ij} = 1$  if  $(i,j)$  is a viable link (0 otherwise). A *spanning relay* is chosen initially as the node of highest degree. A link is then established between it and all adjacent nodes. From the nodes currently spanned, a new spanning relay is selected, adjacent to the maximum number of unspanned nodes, and the process is repeated. It may also be observed from Figures 2 and 3 that MAB solutions generally result in shorter paths than MST solutions, and Figure 4 shows that the difference increases with problem size. However, the add algorithm is still essentially greedy and consequentially crude: better results are to be achieved from TS and SA.

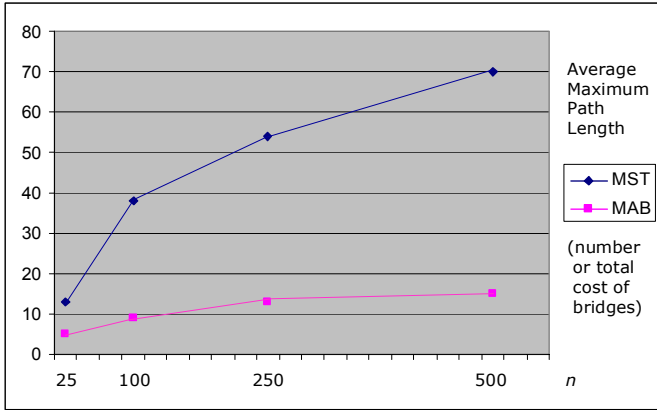


Figure 4. Average maximum path lengths

Figure 5 outlines the TS algorithm for the MAB. It extends ADD's capability using multi-starts. A *Restricted Candidate List (RCL)* is produced during each pass, holding a set of highly-ranked nodes for each construction step. The RCL is used to find unexploited candidates to be included in subsequent passes.

Candidates are stored in a dynamically-created tree, together with the iteration in which they are going to be used. The tree is organized so that parent alterations occur earlier in the construction process than their children. Each node of the tree produces a construction pass, imposing its own alteration along with its preceding ancestor alterations.

The function *BuildTree* selects the highest-degree bridge not marked by the *GlobalTabuVector* to be chosen at the first construction step. *CreateChildren* searches the RCL for the  $k$  most attractive alterations not marked by the local tabu vector, which contains the union of bridges selected by the most recent pass and its ancestor passes. *MaxDepth* and  $k$  specify tree dimensions and *MaxRestarts* defines the number of first-iteration alterations.  $f(s)$  returns the number of bridges for a feasible solution and  $\infty$  otherwise.  $\delta$  is used to limit execution time by further restricting the search neighbourhood.

## TS

```

GlobalTabuVector  $\leftarrow \hat{s} \leftarrow s \leftarrow \emptyset$ 
TreePtr  $\leftarrow$  BuildTree(GlobalTabuVector)
while not TreePtr=NULL do {
  AltList  $\leftarrow$  GetAlts(TreeNode)
   $s \leftarrow$  Add(AltList) // update RCL
  if NodeDepth<MaxDepth and  $f(s) < f(\hat{s}) + \delta$  then
    CreateChildren(RCL, TreePtr, k)
  if  $f(s) < f(\hat{s})$  then {
     $\hat{s} \leftarrow s$ 
    GlobalTabuVector  $\leftarrow$  GlobalTabuVector  $\cup s$  }
  else if NodeDepth = 1 then
    GlobalTabuVector  $\leftarrow$  GlobalTabuVector  $\cup s$ 
  TreePtr  $\leftarrow$  FindNextNode(TreePtr) // depth-
  if TreePtr=NULL // first search
    and Restarts<MaxRestarts then
      TreePtr  $\leftarrow$  BuildTree(GlobalTabuVector) }

```

Figure 5. Tabu-Search for approximating the minimum-bridge solution

An SA approach to the MAB (and related) problems is also under investigation. This uses only node swaps to explore the search space. Disconnected nodes and components are penalized by the evaluation function – but not repaired. This allows the algorithm to cross infeasible regions of the search space, which is essential as large-scale perturbations are normally required to improve upon local ADD solutions. Preliminary results are good. However, this algorithm is only just out of its developmental stage. At the time of publication, fine-tuning is still required and final results will be reported elsewhere.

## IV. RESULTS

This section discusses the performance of the TS algorithm as a relationship between accuracy and run-time. Table I summarizes results from the ADD and TS algorithms for  $n = 30, 100, 300$  &  $1,000$  – 50 runs for each with an RCL length of 4, *MaxDepth* =  $k = 4$ , *MaxRestarts* = 9 and  $\delta = 2$ .

TABLE I. A COMPARISON OF ADD AND TS

$n$	Mean percentage improvement of TS over ADD			
		LOS = 0.5	LOS = 0.7	LOS = 0.9
30	MTD = 0.3	XXX	3.28	4.44
	MTD = 0.5	6.25	7.73	9.04
	MTD = 0.7	6.25	4.48	2.02
		LOS = 0.5	LOS = 0.7	LOS = 0.9
100	MTD = 0.3	9.14	10.76	9.22
	MTD = 0.4	12.29	12.03	15.94
	MTD = 0.5	10.08	14.48	10.70
		LOS = 0.2	LOS = 0.3	LOS = 0.4
500	MTD = 0.2	4.41	5.65	6.95
	MTD = 0.3	5.76	7.76	8.09
	MTD = 0.4	6.43	8.73	10.68
		LOS = 0.15	LOS = 0.2	LOS = 0.25
1,000	MTD = 0.15	3.10	3.82	4.18
	MTD = 0.2	3.83	4.21	3.97
	MTD = 0.25	4.23	5.02	5.05

XXX: No feasible solutions

LOS and MTD are the *line-of-sight probability* and *maximum transmission distance* (relative to a unit square

containing all nodes) respectively, used to generate the edge viabilities  $e_{ij}$ . (An edge is viable if there is LOS and the distance between them is less than MTD.)  $b_j = 1$  for all  $j$  – all nodes are viable bridges in these tests. Table I shows some variance in the relative performance although the general improvement of TS over ADD is clear.

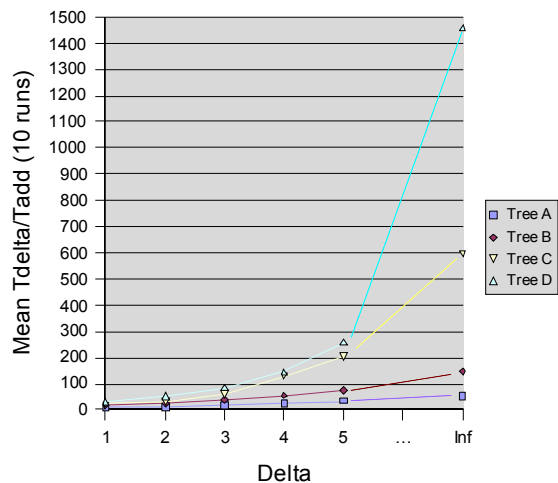


Figure 6. Comparing ADD and TS execution times

Execution time for TS is approximately  $|TreeNodes| \times T_{add}$  where  $T_{add}$  is the time taken by ADD and  $|TreeNodes|$  is the total number of tree nodes created during the search. In the unrestricted case ( $\delta = \infty$ ),  $|TreeNodes|$  can be calculated easily based on values for  $k$ ,  $MaxDepth$  and  $MaxStarts$ . It remains to consider the effect of lower values of  $\delta$  on execution time and accuracy.

TABLE II. SEARCH TREES USED TO COMPARE ADD AND TS

Tree	$k$	$MaxDepth$	$MaxStarts$
A	3	3	5
B	4	3	9
C	4	4	9
D	5	4	12

Define  $T_{delta}$  to be the time taken by TS with a given  $\delta$ . Figure 6 shows mean values of  $T_{delta}/T_{add}$  for four different sets of search tree dimensions (given in Table II) over problem instances with  $n = 500$ ,  $MTD = LOS = 0.2$ . Lower values of  $\delta$  reduce execution time significantly compared with the unrestricted case. Figure 7 shows that there is little difference in solution quality between  $\delta = \infty$  and  $\delta = 5$  and only moderate deterioration by  $\delta = 3$ . This confirms that lowering  $\delta$  reduces execution time considerably while still returning good results.

## V. CONCLUSIONS

Obviously there is *some* expense to greater accuracy. Run-times increase with more sophisticated search parameters and larger search neighbourhoods. The ideal balance will depend on the application being considered. Longer run-times may well be tolerated in static design problems such as wireless

broadband distribution networks [8] and will yield the closest to optimal solutions. In dynamic situations, such as the frequent re-calculation of bridge spanning trees in response to link/node failure, quicker, less-precise searches will be necessary. Fortunately, the accuracy-complexity trade-off of TS appears favourable in this respect.

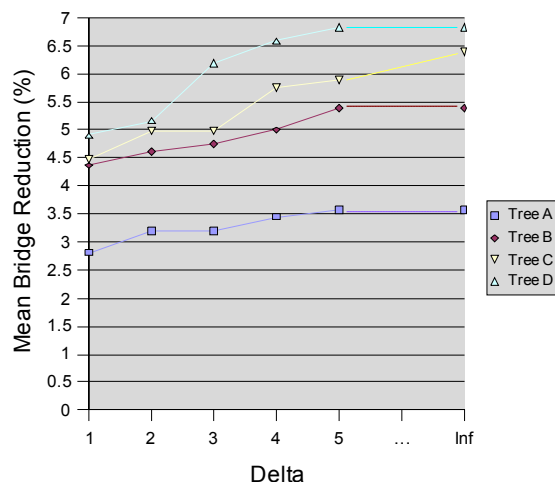


Figure 7. Comparing ADD and TS accuracy

A final consideration must be given to the distribution of these processes. The versions of ADD, TS and SA discussed here are centralized (they take a global view of the network problem/solution), which is appropriate for static network design problems. For dynamic bridging protocols, however, the algorithms will be required to run independently on individual nodes [9] and distributed versions of ADD, TS and SA have to be found. This is the next stage of the research.

## REFERENCES

- [1] B.Y. Wu and K.-M. Chao, *Spanning Trees and Optimization Problems*, Chapman & Hall, 2004.
- [2] J.B. Kruskal, "On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem", *Proc. American Math. Soc.*, vol. 7, pp48-50, 1956.
- [3] E.W. Dijkstra, "A Note on Two Problems in Connexion with Graphs", *Numerische Mathematik*, vol. 1, pp269-271, 1959.
- [4] M.R. Garey and D.S. Johnson, *Computers and Intractability: A guide to the theory of NP-Completeness*, Freeman, 1979.
- [5] V. Grout, "Principles of Cost Minimisation in Wireless Networks", *J. Heuristics*, vol. 11, no. 2, pp115-133, 2005.
- [6] V. Grout, "Ne'er the Twain Shall Meet: Bridging the network optimisation reality gap", *Mathematics Today*, vol. 41, no. 5, pp157-160, October 2005.
- [7] E. Aarts and J.K. Lenstra, *Local Search in Combinatorial Optimisation*, Princeton University Press, 2003.
- [8] Fowler, T., "Mesh Networks for Broadband Access", *IEE Review*, pp17-22, January 2000.
- [9] Cisco Systems, "Spanning Tree Protocol", <http://www.cisco.com/warp/public/473/17.html>, 27<sup>th</sup> January 2006.