PRIFYSGOL

glyndŵr
UNIVERSITY

Glyndŵr University
Glyndŵr University Research Online

Computing

Computer Science

7-1-2006

Optimisation Techniques for Wireless Networks

Mike J. Morgan
m.j.morgan@glyndwr.ac.uk

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

Follow this and additional works at: http://epubs.glyndwr.ac.uk/cair

Part of the Computer and Systems Architecture Commons, Digital Communications and Networking Commons, Hardware Systems Commons, and the Systems and Communications Commons

Recommended Citation

Optimisation Techniques for Wireless Networks

Abstract

This paper introduces two new algorithms for the minimum connected dominating set problem. The problem and its relevance to various aspects of wireless network optimisation are briefly outlined followed by a description of the suggested techniques. Results show that these algorithms outperform a number of previous approaches in terms of solution quality and potential for future work is discussed.

Keywords

Wireless network design, Simulated annealing, Tabu search

Disciplines

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

Comments

Optimisation Techniques for Wireless Networks

Mike Morgan and Vic Grout

Centre for Applied Internet Research (CAIR), University of Wales, NEWI, Wrexham, UK
{mi.morgan|v.grout}@newi.ac.uk

Abstract

This paper introduces two new algorithms for the minimum connected dominating set problem. The problem and its relevance to various aspects of wireless network optimisation are briefly outlined followed by a description of the suggested techniques. Results show that these algorithms outperform a number of previous approaches in terms of solution quality and potential for future work is discussed.

Keywords

Wireless network design, Simulated annealing, Tabu search

1. Introduction

The minimum connected dominating set problem (MCDS) arises in many practical networking scenarios including design and virtual backbone organisation. This paper proposes two new centralised algorithms for the problem, based on *Tabu Search* (*TS*) (Glover and Laguna, 1997) and *Simulated Annealing* (*SA*) (Aarts and Korst, 1989) principles. They are shown to outperform the greedy heuristics of Guha and Khuller (1998) and Grout (2005).

2. Minimum Connected Dominating Sets

In practical terms, the MCDS problem may be thought of as minimising the number of backbone nodes in a network, where each node must either be a backbone node or be adjacent to one. Formally, a MCDS of a graph G = (V,E) is a subset V' ⊆ V such that all nodes in V-V' are neighbours of nodes in V', V' forms a connected component of G and |V'| is a minimum. The problem is NP-Complete (Garey and Johnson, 1979). Figure 1 shows an example.
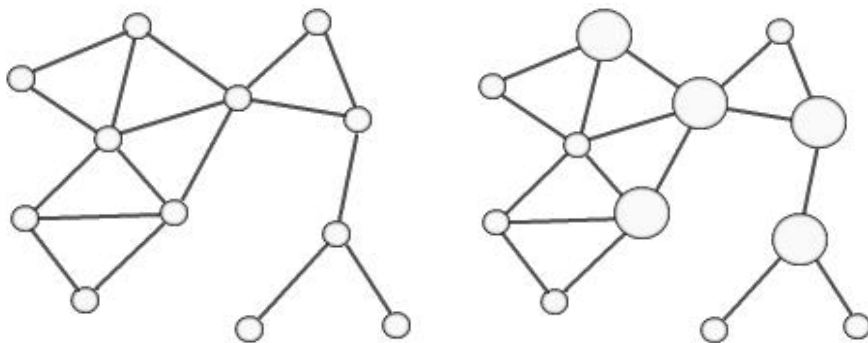

Figure 1: **An Example Network with MCDS Nodes Enlarged**

A practical application involves minimising the number of relay nodes in a wireless multihop network. This has been tackled dynamically for ad-hoc and sensor networks e.g. (Alzoubi *et al.*, 2002) (Dai and Wu, 2004), but it also has static applications in fixed wireless network design. The latter problem is formulated by Grout (2005) with consideration of practical constraints. These are classified as *add* or *drop* constraints, which may be violated by adding

or dropping a node, respectively. In this paper, add constraints are considered in isolation, whilst drop constraints are set aside for future work.

Add constraints come in two forms: edge and node viability. An edge is only viable between two sites if it is possible to transmit data between them. They are easily handled if the problem graph is defined only to include viable edges. A node is viable if its corresponding site is able to house relay equipment. These can be handled using a boolean vector to mark viable nodes. It may be observed that, in the absence of both edge and node viability constraints, the problem is solved trivially by a star/hub and spoke subgraph.

A number of centralised heuristics have been proposed. Two common approaches are to add nodes to the solution one at a time, choosing at each stage the node or combination of nodes which connects up the largest remaining portion of the network (Grout, 2005) (Guha and Khuller, 1998), or to form a maximal independent set and then add nodes to ensure connectivity (Alzoubi *et al.*, 2002). A third approach begins with all viable nodes and drops components, examples being the drop algorithm by Grout (2005) which drops edges and a similar algorithm by Butenko et al. (2003) which drops nodes. However, these algorithms perform fairly poorly when compared with other approaches and their main application would be ensuring connectivity at all times during distributed processes or in handling drop constraints.

In contrast to the methods outlined above, this paper uses two meta-heuristic approaches. Simulated annealing (Aarts and Korst, 1989) is an adaptation of local search equipped with the ability to escape local optima. Here its implementation is unusual as it uses a modified evaluation function to penalise infeasible solutions. The technique is common to evolutionary algorithms but rarely used with trajectory based techniques such as SA (Hoos and Stutzle, 2005). TS (Glover and Laguna, 1997) is another meta-heuristic which uses memory to learn from previous solutions. The algorithm in section 4 adapts a construction algorithm to perform *strategic oscillation* in a manner similar to that outlined by Glover (2000).

3. MCDS/SA: a Simulated Annealing Algorithm for MCDS

With regard to local search, it is difficult to improve upon results from existing greedy algorithms as they are often locally optimal with respect to any easily-defined neighbourhood. The problem's objective function $f(s)$ returns the number of relays if a candidate solution is feasible and infinity otherwise; it can only be improved by a move which drops more relays than it adds (maintaining feasibility), or makes the solution feasible when it was not.

The MCDS/SA algorithm gets around this difficulty in two ways: firstly, by probabilistically accepting non-improving solutions in a manner common to all SA algorithms and secondly, by using a modified evaluation function $f'(s)$. This modified function differentiates between solutions with equal $f(s)$ values, and penalises (rather than ruling out) infeasible solutions. The *temperature* parameter controls the probability with which non-improving solutions are accepted and is reduced during search to lead the algorithm back to feasibility.

It is necessary to introduce another solution property to consider $f'(s)$ in detail. The *coverage* of a node is defined to be the number of relays in its neighbourhood. The *total coverage C* of the network is equal to the sum of the coverages of every node of the graph. The theory is that the greater the total coverage, the more likely it is that a relay can be dropped without affecting feasibility. So the value of C is subtracted from $f'(s)$ to favour solutions with higher coverage. The number of relays is multiplied by the maximum total coverage *2m* to compensate for this, where m is the number of edges of the problem graph.

Infeasible solutions fall into two types. A solution may have disconnected nodes (nodes with

no adjacent relays) or disconnected components. As it is necessary to penalise both of these categories, the evaluation function *f'(s)* is outlined as follows:

$$f'(s) = 2m(ny+2z+|s|)-C$$

Where *y* is the number of disconnected components, *z* is the number of disconnected nodes and *|s|* is the number of relays in the solution. This function is intended to penalise where necessary whilst ensuring that the minimum value for *f'(s)* would always give a minimum for *f(s)*.

Three moves are used by the algorithm: adding a relay, dropping a relay and a combination of the two. The combination move is preferred, with a 10% probability of not adding or not dropping at each iteration. If a node is dropped without creating an infeasible solution, it will be dropped without adding anything back to encourage minimisation of the CDS. Once a move is performed, values for *z*, *|s|* and *C* can be recalculated in $O(d(v))$ time, where $d(v)$ is the degree of the vertex altered. The value of *y* presents more of a challenge as it would appear to require a breadth first search for every move, which would slow the algorithm down considerably. However, by organising the moves into systematic order it is possible to reduce the frequency with which breadth-first search is required.
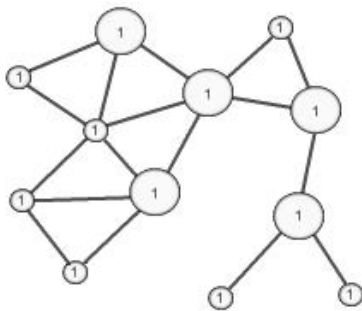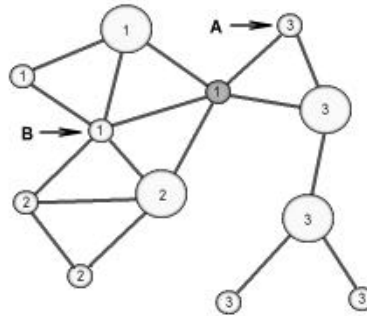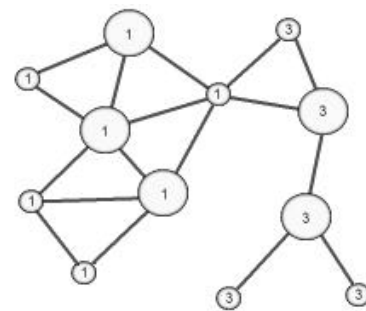

| Figure 2a | Figure 2b | Figure 2c |

A component identifier (ID) is maintained for each node (Figure 2 shows an example). Disconnected nodes will be given a default ID. Figure 2a shows a fully connected network. Once a node is dropped, a modified breadth first search (BFS) must be performed for all components and their IDs updated (Figure 2b). At this point, add moves may be evaluated by examining the number of neighbouring relays with different IDs. In figure 2b, the node marked A will not reduce *y*, whilst node B will reduce *y* by 1. If node B is added back, the IDs can be relabelled in linear time. If a disconnected node is added, a new component is created and all its neighbours given the new ID. Therefore, it is possible to evaluate up to *n*-1 combination moves between breadth first searches, provided that they all drop the same node.

The pseudocode in figure 3 shows how MCDS/SA takes advantage of this. The algorithm iterates through relay nodes, dropping each one tentatively before defining a candidate list of non-relay nodes to add back. The move corresponding to each candidate node is evaluated incrementally as illustrated in figure 2. (Note that there is a ten percent chance of not adding or not dropping each time and evalations must be made accordingly.) If the new solution is accepted, the algorithm immediately proceeds to drop the next relay as both candidate lists and ID vectors are invalidated at this point.

Moves are accepted or rejected in accordance with the *metropolis criterion*, which stipulates that every improving move will be accepted and that non-improving moves will be only be accepted if a random variable [0,1) is less than $e^{-\Delta_{eval}/T}$, where Δ_{eval} is the difference in *f'(s)* for the old and new solutions and *T* is the current temperature. The temperature is periodically reduced by some multiplying factor *a* in the range [0,1).

```
procedure MCDS/SA
        initialise temperature T and solution s
        create component ID vector cid for s   %BFS
        while (further improvements found)
                for all relays r   %begin pass
                        s' ← s - r
                        create component ID vector cid'for s' %BFS
                        if (s'is feasible) then
                                continue   %proceed to next relay
                        end-if
                        generate a candidate list of non-relay nodes to become relays
                        for all nodes l in candidate list
                                randomly choose type of move (add/drop/both)
                                if (move type = add) then
                                        s'' ← s + l
                                        evaluate move (s,cid,g)
                                else if (move type = drop) then
                                        s'' ← s'
                                        evaluate s'' against s
                                else if (move type = both) then
                                        s'' ← s' + l
                                        evaluate move (s',cid',g)
                                end-if
                                if (move accepted) then   %metropolis criterion
                                        relabel cid
                                        s ← s''
                                        break   %proceed to next relay
                                end-if
                        end-for
                end-for   %end pass
                T ← aT
        end-while
end-procedure
```

**Figure 3: Pseudocode for MCDS/SA**

The strategy for defining candidate lists is as follows. Lists have constant length (a figure of 20 was used in tests). They are initially filled with promising two-hop neighbours, with any remaining space populated randomly. Figure 4 gives an example. 4a shows a portion of a network. Once a relay is dropped (marked grey in figure 4b), several nodes become disconnected (marked black). The adjacency list of each black node is scanned for potential new relays (marked by arrows in figure 4c). Of these, the relay marked A can connect three black nodes and is considered the best new candidate. The list will be populated with the best candidates in order, followed by random candidates to fill up space where necessary.



**Figure 4a**          **Figure 4b**          **Figure 4c**
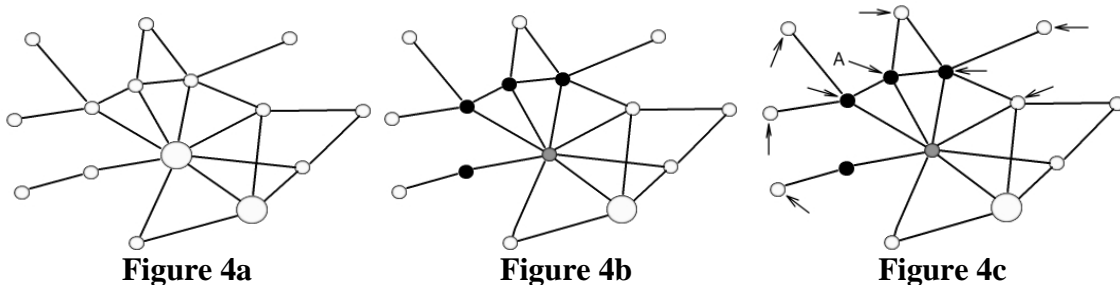
## 4. MCDS/TS: A Tabu Search Algorithm for MCDS

The last section shows that local search can be complex for this problem. The best-first approach normally used in standard tabu search (TS) algorithms would introduce a minimum complexity of O($nm$) per move, even if all the efficiency improvements from MCDS/SA were

used. It is possible that efficiency could be improved a little with more candidate lists or path-relinking, but the process would still be time-consuming.

An alternative is to use multi-starts to modify an existing greedy heuristic. This process is common to more than one meta-heuristic, other examples being GRASP and Adaptive Iterated Construction Search (AICS). MCDS/TS modifies the *add* algorithm (Grout, 2005) to perform strategic oscillation in a manner based on work by Glover (2000).

The add algorithm is an efficient heuristic which produces good results in practice. To begin, every node is marked disconnected. The node with the highest degree is added to the dominating set, and its neighbours are marked as dominated. For each subsequent iteration, the dominated node with the most disconnected neighbours (or highest *yield*) is added to the set and the algorithm terminates when no more disconnected nodes can be found.

This algorithm is modified by the use of multi starts. A *restricted candidate list (RCL)* is maintained for each step, holding a set of highly ranked candidates. The RCL is used to find unexploited candidates to be included in subsequent passes. For example, on a given step of the add algorithm, the RCL may record several nodes with yields equal to the highest or thereabouts. Choosing any one of these may have a profound effect on the subsequent behaviour of the algorithm. The extreme case changes the first node selected, which may entirely alter the list of dominated nodes for several steps.

In fact, the result of changing many nodes generally results in poor solutions, but by changing one node per pass in systematic order good quality solutions may be obtained. This is achieved by storing alterations in a dynamically updated tree. Each alteration consists of an integer pair (iteration, node). The tree is arranged in such a manner as each alteration occurs later than its parent. For every alteration node, a corresponding pass is produced, imposing its own alteration and the alterations of each of its ancestors by reading back up the tree.
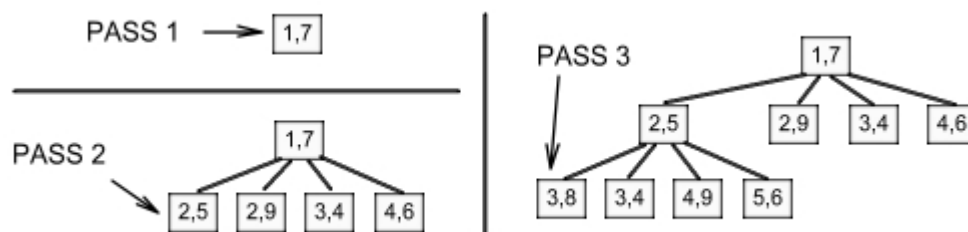


**Figure 5: A Tree Example**

Figure 5 shows an example. After each pass, RCLs are inspected to select the 4 unexploited nodes with highest yield. They are made into children of the current alteration. This is repeated, up to a depth of *maxDepth*, traversing the tree in depth-first order. Once the traversal is complete, a new tree is built up to a limit of *maxStarts* trees. In the interests of efficiency, the tree may be pruned by stipulating that only passes producing solutions within a given value **d** of the best-known solution may have children. Pseudocode is given in figure 6.

The function *BuildTree* selects the highest degree node not marked by the *GlobalTabuVector* to be chosen at the first construction step. The value $k$ defines the number of children per tree node and the function *GetAlterations* reads back up the tree from the current node to acquire all the alterations it needs to impose. *CreateChildren* selects the $k$ best alterations, whose nodes are not contained in the *local tabu vector* (the union of nodes used in the current pass and all ancestor passes). The objective function *f(s)* returns the size of the dominating set for a feasible solution and infinity otherwise.

```
procedure MCDS/TS
        GlobalTabuVector ← ŝ ← s ← ∅
        TreePtr ← BuildTree(GlobalTabuVector)
        while (not TreePtr = NULL) do
                AlterationList ← GetAlterations(TreeNode)
                s ← Add(AlterationList)              % updates RCL
                if (NodeDepth<MaxDepth and f(s) < f(ŝ)+d ) then
                        CreateChildren(RCL, TreePtr, k)
                end-if
                if ( f(s) < f(ŝ) ) then
                        ŝ ← s
                        GlobalTabuVector ← GlobalTabuVector ∪ s
                else if (NodeDepth = 1) then
                        GlobalTabuVector ← GlobalTabuVector ∪ s
                end-if
                TreePtr ← FindNextNode(TreePtr)      % depth first search
                if(TreePtr = NULL and Restarts < MaxStarts) then
                        TreePtr ← BuildTree(GlobalTabuVector)
                end-if
        end-while
        return ŝ
end-procedure
```

Figure 6: **Pseudocode for MCDS/TS**

5. Results

For testing, nodes were placed randomly within a unit square. A maximum transmission distance MTD and line of sight probability LOS were used to define edge viabilities, whereby edges connected nodes less than MTD apart with probability LOS. A value IN was defined as the probability of any node being infeasible.

The three algorithms used for comparison were the add algorithm and Guha and Khuller's two algorithms, hereafter referred to as GK1 and GK2. The metaheuristics outperform these algorithms as expected, given that both TS ans SA methods conduct a longer search.
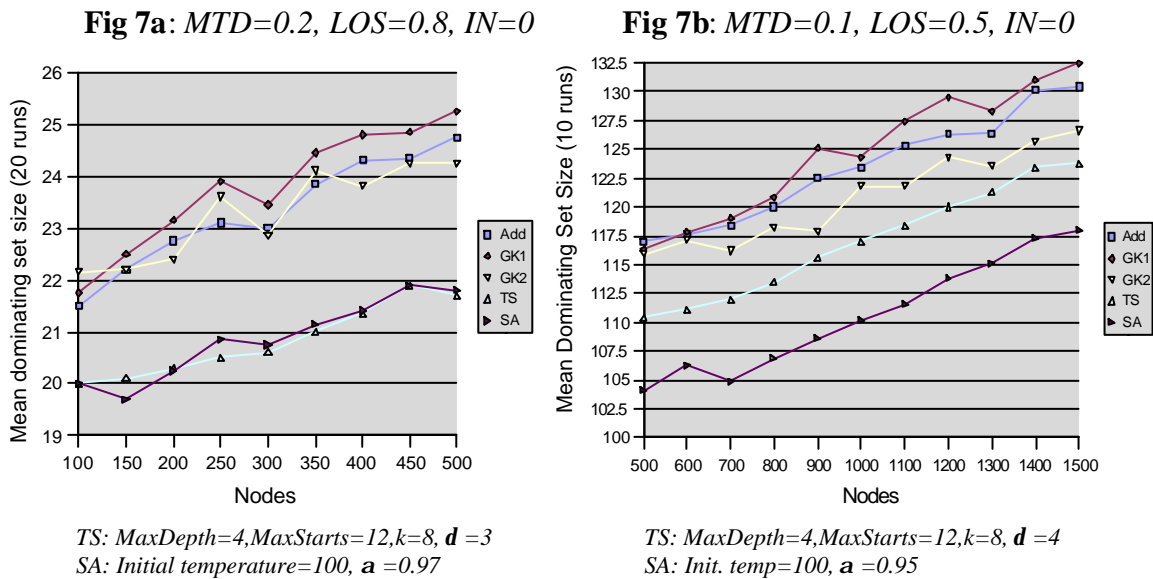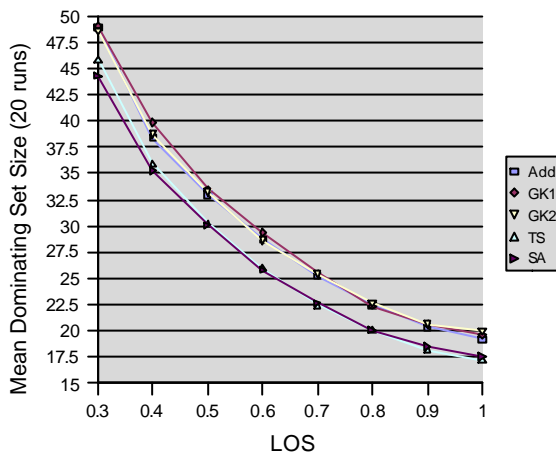
Fig 7a: *MTD=0.2, LOS=0.8, IN=0*   **Fig 7b**: *MTD=0.1, LOS=0.5, IN=0*

TS: MaxDepth=4,MaxStarts=12,k=8, d =3   *TS: MaxDepth=4,MaxStarts=12,k=8, d =4*
SA: Initial temperature=100, a =0.97   *SA: Init. temp=100, a =0.95*

Figure 7: Plots of dominating set size against *n* for constant MTD, IN and LOS

Figure 7 shows the effect of increasing *n* on algorithm performance. Whilst there is little difference between SA and TS for smaller graphs (Figure 7a), SA outperforms TS on larger ones (Figure 7b). Comparing figures 7a and b, 7a shows TS giving better solutions for n=500 with MTD = 0.2 and LOS = 0.8, while SA is far more successful with MTD = 0.1 and LOS = 0.5 in 7b. This shows that edge density also has a bearing on the two algorithms' relative performance. Figure 8 confirms this, showing the result of varying LOS for constant *n* and MTD. In figure 8a TS begins to outperform SA as the density increases and although SA is superior in figure 8b a relative improvement can be observed for TS. The effect of increasing edge density explains the inconcistency in results for figure 7. Comparing 7a and 7b, it is clear that SA is more successful for larger *n*, but this is not noticeable on considering either chart in isolation. This is explained by the fact that edge density increases with increasing *n*.

Fig 8a: *n=200, MTD=0.2, IN=0*   **Fig 8b:** *n=1000, MTD=0.1, IN=0*
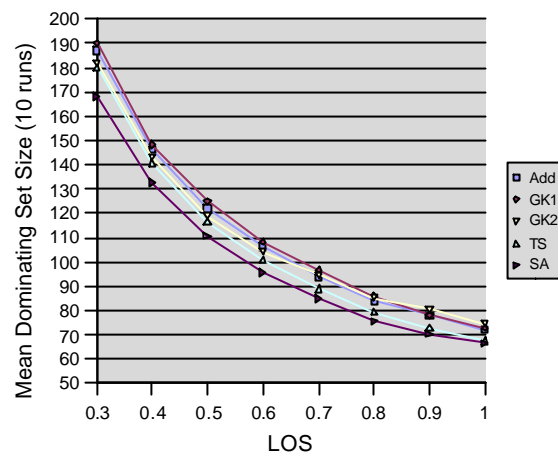
TS: MaxDepth=4,MaxStarts=12,k=8, **d** =3   TS: MaxDepth=4,MaxStarts=12,k=8, **d** =4
SA: Initial temperature=100, **a** =0.97   SA: Initial temperature=100, **a** =0.95

Figure 8: Plots of dominating set size against LOS for constant MTD, IN and *n*.

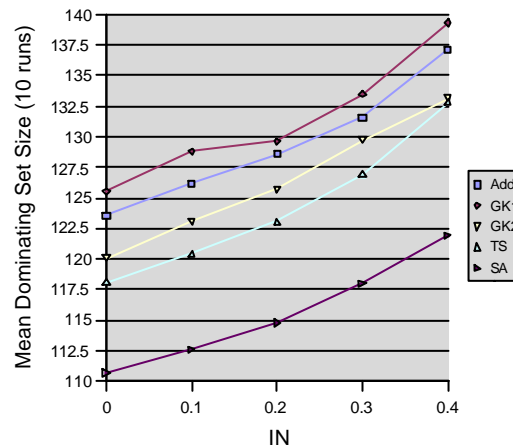Whilst figures 7 and 8 deal exclusively with edge constraints (IN=0), the effect of increasing IN is observed in figure 9. In this case the results are a little less consistent. SA may appear to improve slightly over TS as IN increases, but the results are by no means conclusive.

Fig 9a: *n=200, MTD=0.2, LOS=0.5*   **Fig 9b:** *n=1000, MTD=0.1, LOS=0.5*
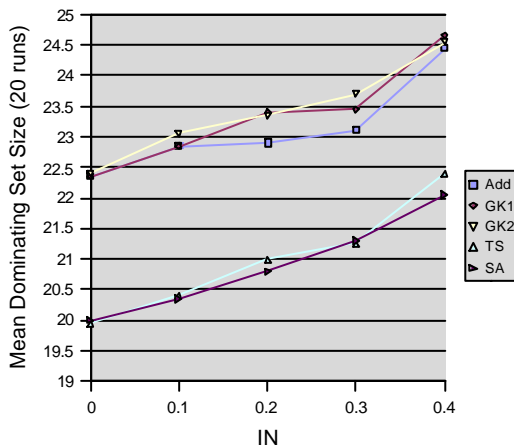
TS: MaxDepth=4,MaxStarts=12,k=8, **d** =3   TS: MaxDepth=4,MaxStarts=12,k=8, **d** =4
SA: Initial temperature=100, **a** =0.97   SA: Initial temperature=100, **a** =0.95

Figure 9: Plots of Dominating set size against IN for constant *n*, MTD and LOS

6. Conclusions

It can be seen from the results that each of these algorithms has its purpose. SA tends to prefer larger, sparser graphs while TS tends to prefer smaller, denser instances and the underlying reason for this may easily be conjectured. The TS algorithm requires that the connectivity constraint be met at every stage of construction which gives it the potential to overlook good relay choices. The extent to which this adversely affects TS performance ought to depend on the proportion of disconnected nodes over the course of the algorithm's pass. This proportion will naturally be greater in large, sparse networks.

This conjecture is backed up by the relative behaviours of add and GK2 in the results section. GK2 (which does not require the connectivity constraint) begins to outperform ADD as the networks get larger (Figure 7), but add comes into its own as the edge density increases (Figure 8). Based on these findings, it may be profitable to consider size, edge density and proportion of inviable nodes before deciding which approach to use. This does not only apply to the SA and TS approaches outlined, but also to simple greedy heuristics like add and GK2, both of which are useful in the event that speed is a requirement. A simple hyper-heuristic might evaluate node count and edge density before applying its chosen method, or a hybrid approach could be sought.

With regard to fixed wireless network design, a beginning has been made. However, the algorithms must be extended to tackle capacitative and redundancy constraints if they are to be of significant practical use. Where virtual backbones, routing and broadcasting in ad-hoc networks are concerned, the potential for a distributed counterpart to one or both of these algorithms would need to be investigated. These would constitute the main body of research work hereafter.

7. References

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

Alzoubi, K. Wan, P.J. and Frieder, O. (2002) "Distributed Construction of Connected Dominating Set in Wireless Ad Hoc Networks", *Proceedings of IEEE INFOCOM,* vol. 3, June 2002, pp. 1597-1604

Butenko, S. Cheng, X. Oliveira, C. and Pardalos, P. (2003) "A new algorithm for connected dominating sets on ad hoc networks", In: Butenko, S. Murphey, R. and Pardalos, P. *Recent Developments in Cooperative Control and Optimization*, *pages 61-73*. Kluwer

Dai, F. and Wu, J. (2004) "An Extended Localized Algorithm for Connected Dominating Set Formation in Ad Hoc Wireless Networks", *IEEE Transaction on Parallel and Distributed Systems,* Vol. 15 No. 10 October 2004 pp. 908-920

Garey, M. and Johnson, S. (1979) *Computers and Intractability, a Guide to the Theory of NP-Completeness,* Freeman

Glover, F. and Laguna, M. (1997) *Tabu Search,* Kluwer Academic Publishers

Glover, F. (2000) "Multi-Start and Strategic Oscillation Methods - Principles to exploit adaptive memory", In: Laguna, M. and Gonzales-Valarde, J. *Computing Tools for Modeling, Optimization and Simulation: Interfaces in Computer Science and Operations Research.* pp 1-24. Kluwer

Grout, V. (2005) "Principles of Cost Minimisation in Wireless Networks", *Journal of Heuristics,* 11: 115-133

Guha, S. and Khuller, S. (1998) "Approximation Algorithms for Connected Dominating Sets", *Algorithmica,* 20: 374-387.

Hoos, H. and Stutzle, T (2005) *Stochastic Local Search Foundations and Applications,* Morgan Kauffman