PRIFYSGOL glyndŵr UNIVERSITY

Glyndŵr University
Glyndŵr University Research Online

Computing | Computer Science

5-1-2007

Metaheuristics for Wireless Network Optimisation

Mike Morgan

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
Morgan, M. & Grout, V. (2007), 'Metaheuristics for Wireless Network Optimisation'. [Paper presented to the 3rd IARIA/IEEE Advanced International Conference on Telecommunications (AICT 2007) 13th -19th May 2007]. Mauritius: IEEE

This Conference Paper is brought to you for free and open access by the Computer Science at Glyndŵr University Research Online. It has been accepted for inclusion in Computing by an authorized administrator of Glyndŵr University Research Online. For more information, please contact d.jepson@glyndwr.ac.uk.

Metaheuristics for Wireless Network Optimisation

Abstract
This paper introduces two new algorithms for the minimum connected dominating set (MCDS) problem with constraints applicable to wireless network design, based on simulated annealing and tabu search principles. Each algorithm is tested on a selection of random graphs and shown to produce significantly smaller connected dominating sets when compared to a number of established methods. The simulated annealing algorithm is found to favour large, sparse graphs while the tabu search heuristic prefers smaller dense instances. In conclusion, we consider the adaptation of these algorithms to hybrid techniques and comment on the possible use of hyper-heuristics.

Keywords
wireless network design, minimum connected dominating set, simulated annealing, tabu search, metaheuristics

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

Comments
Copyright © 2007 IEEE – All Rights reserved. This paper was presented to the 3rd IARIA/IEEE Advanced International Conference on Telecommunications (AICT 2007) 13th -19th May 2007 in Mauritius. The proceedings were published by the IEEE and are available at http://ieeexplore.ieee.org This material is posted here with permission of the IEEE and the author. Such permission of the IEEE does not in any way imply IEEE endorsement of any of the products or services of Glyndwr University Wrexham. Internal or personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution must be obtained from the IEEE by writing to pubs-permissions@ieee.org. By choosing to view this document, you agree to all provisions of the copyright laws protecting it.

This conference paper is available at Glyndŵr University Research Online: http://epubs.glyndwr.ac.uk/cair/63

Metaheuristics for Wireless Network Optimisation

Mike Morgan, 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 (MCDS) problem with constraints applicable to wireless network design, based on simulated annealing and tabu search principles. Each algorithm is tested on a selection of random graphs and shown to produce significantly smaller connected dominating sets when compared to a number of established methods. The simulated annealing algorithm is found to favour large, sparse graphs while the tabu search heuristic prefers smaller dense instances. In conclusion, we consider the adaptation of these algorithms to hybrid techniques and comment on the possible use of hyper-heuristics.**

Keywords; wireless network design; minimum connected dominating set; simulated annealing; tabu search; metaheuristics.

I. INTRODUCTION AND RELATED WORK

A *Minimum Connected Dominating Set* (*MCDS*) of a graph G=(V,E) is a subset $S \subseteq V$ such that each node in $V \setminus S$ is adjacent to one or more nodes in V, the subgraph G' induced by *S* is a connected component of G and $|S|$ is a minimum. The problem is known to be NP-Complete [6]. The practical application of the MCDS problem in wireless network design arises from the need to minimise the number of relays (switches, etc.) in a given network. It forms an integral part of the design problem for static networks along with self-organisation problems in mobile ad-hoc networks or MANETs. The former may be tackled using centralised methods, while the latter relies on distributed approaches. This paper proposes two new centralised algorithms for the static problem variant.

A fairly complex family of constraints has been outlined for the design problem in [9]. In this paper, a reduced version of this constraint set is considered in order to make a beginning. The simplified model involves defining a set of feasible edges based on maximum transmission distances and line-of-sight considerations, whilst constraining certain nodes so that they may not join the dominating set. This deals with the case in which certain premises may not be able to house relay equipment, whilst also addressing the transmission limitations of various wireless technologies. More complex redundancy and capacitative constraints are set aside for future work.

The two algorithms are each implementations of well known metaheuristics: simulated annealing [1] and tabu search [7]. The SA algorithm implements a modified evaluation function which introduces a slight gradient into the extensive plateaux commonly found in the problem's search space. This function can be used with a number of local search methods besides SA and extended to tackle the generalised connected dominating set problem [11] with a simple alteration, if required. The tabu search algorithm is more esoteric and can be considered a semi-greedy metaheuristic in its own right. It is a deterministic approach inspired by techniques in [8] whereby multi-starts, as used in semi-greedy methods, are defined as an extreme case of one-sided strategic oscillation. This principle is employed to create an effective construction search based on a simple greedy algorithm. The algorithms are seen to outperform simpler heuristics and shown to have individual strengths and weaknesses, pointing to the possible use of hyper-heuristics or hybrid techniques for the MCDS problem.

To date, much of the work on connected dominating sets has focused on distributed methods for virtual backbones in ad hoc networks [5,13]. Whilst some work has been carried out on centralised approximation algorithms, these are generally simple, greedy heuristics. Algorithms such as those found in [9] and [10] use the greedy criterion whereby the node or combination of nodes which connect up the largest remaining portion of the network are added to *S* at each iteration, requiring the subgraph induced by *S* to be connected at each stage or not, as the case may be. Other methods, such as [2] involve forming a maximal independent set and then adding nodes to form a CDS. This approach yields a constant approximation ratio, but results are relatively poor in practice [5]. A final technique is to commence with all nodes in an initial dominating set and drop nodes or edges at each stage of the algorithm. Examples include [3] and [9]. Again, these perform poorly in practice and their main use is in handling complex constraints or in ensuring connectivity at all times during distributed processes. Whilst simulated annealing and tabu search have been used successfully in network design problems such as the capacitative minimum spanning tree [4], we are aware of no reference to published work on their use with this problem. These methods conduct far lengthier searches than the simple, greedy approaches outlined above and produce better results as a consequence.

II. THE MCDS/SA ALGORITHM

The simulated annealing algorithm for MCDS (*MCDS/SA*) is outlined in the sections that follow. Initially we outline the modified evaluation function $f'(S)$, before defining the search

neighbourhood with regard to relevant efficiency considerations.

A. The Modified Evaluation Function

The unmodified objective function $f(S)$ for the MCDS problem is as follows:

$$f(S) = \begin{cases} |S| : S \text{ is a CDS of G} \\ \infty \ : \text{otherwise} \end{cases}. \quad (1)$$

It is difficult to produce an effective perturbation search using this as an evaluation function; greedy algorithms will typically produce solutions which are locally optimal with respect to any easily defined neighbourhood. The only way to improve a solution would be to reduce the number of relays whilst maintaining feasibility or to make a solution feasible when it was not. It is difficult to concieve a simple perturbation which is likely to achieve this on a regular basis. The search may be led along extensive plateaux, neither improving the solution nor terminating at a strict local minimum or it may terminate immediately with no improvement, depending on the acceptance criterion used.

The use of SA will combat this to some extent by probabilistically allowing non-improving solutions. However, with a modification to the evaluation function, the situation can be improved further. Define the *coverage C* of a solution to be the sum of the degrees of all its relay nodes. The greater the coverage of the solution, the more likely it is that a relay can be dropped without affecting feasibility. This is because an increase in total relay degree will result in non-relay nodes being adjacent to a greater number of relays on average. Therefore, if the evaluation function is altered to maximise C for sets of equal size, it becomes possible to introduce a gradient into the plateaux and to guide the search process in an effective manner. Additionally, by applying penalties to infeasible solutions rather than ruling them out entirely, the search trajectory can cross infeasible regions of the solution space, temporarily accepting solutions with more than one connected component or with one or more disconnected nodes in the interests of diversification at higher temperatures. The modified evaluation function, $f'(S)$, is:

$$f'(S) = \begin{cases} 2m(ng + |S| + 2z) - C : g > 0 \\ \infty \ : \text{otherwise} \end{cases}, \quad (2)$$

where g is the number of connected components of the subgraph induced by S, z is the number of nodes in V-S with no adjacent relay, C is the coverage of the solution and n and m are equal to |V| and |E|, respectively. A minimum value for $f'(S)$ will always give a minimum for $f(S)$.

B. Search Neighbourhood Definition and Efficiency

The search neighbourhood is a simple variant on 2-node swaps: one node is added to S and another is removed followed by a re-evaluation of $f'(S)$. However, in order to change the size of the CDS, two alterations have been made to this simple neighbourhood definition. Firstly, we probabilistically introduce moves which do not add or do not drop with a given probability P_{move} and secondly, whenever we find a drop move which produces a feasible solution, we do not add anything back to S.

```
procedure MCDS/SA
   initialise temperature T and solution S
   create ID vector id for S                          // BFS
   evaluate f'(S)
   S_best ← S
   while (there is a change in f'(S))
      for all relays u
         S' ← S - u
         create ID vector id' for S'                  // BFS
         evaluate f'(S')
         if (S' is feasible) then
            S ← S'
            if (f(S_best)<f(S)) then
               S_best ← S
            end-if
            continue                 // proceed to next relay
         end-if
         generate candidate list
         for all nodes v in candidate list
            randomly choose type of move (add/drop/swap)
            if (move type = add) then
               S'' ← S + v
               Δf'(S) ←f'(S'')-f'(S)
            else if (move type = drop) then
               S'' ← S'
               Δf'(S) ←f'(S'')-f'(S)
            else if (move type = swap) then
               S'' ← S' + v
               Δf'(S) ←f'(S'')-f'(S')
            end-if
            if (Δf'(S) < 0 or random[0,1) < e^(-Δf(S)/T)) then
               relabel id
               S ← S''
               if (f(S_best)<f(S)) then
                  S_best ← S
               end-if
               break                 // proceed to next relay
            end-if
         end-for
      end-for
      T ← aT
   end-while
   return S_best
end-procedure
```

Figure 1. Pseudocode for MCDS/SA

By contrast to the random sampling used in many SA algorithms, MCDS/SA samples the search neighbourhood in a strict order, subject to the probabilistic elements already described. This order is designed to reduce the need for breadth-first search (BFS) to evaluate g, exploiting that fact that BFS is only required when a node is dropped, as long as a list of component identifiers (*ID*s) is kept for each node of the graph. For each relay node u, we calculate $f'(S)$ and $f'(S\text{-}u)$ and create ID lists for both S and $S\text{-}u$ using BFS. If $S\text{-}u$ is feasible, we drop u from S and proceed to the next relay. Otherwise we produce a candidate list of non-relay nodes v to add back. For each node v, we select a move (swap, add only or drop only). If the drop only move is selected, no further recalculation is necessary. Otherwise, v's neighbour list is scanned for relays with IDs different from v's, using the IDs for $S\text{-}u$ if swapping or the IDs for S if adding only. If v has no neighbouring relays, a new component has been created and g will be increased by one. Otherwise the resulting value for g is reduced by the number of unique relay IDs not equal to v's. Recalculation of z and C is carried out incrementally. A relay's degree is added to

or subtracted from C when it is added or dropped from S and z is updated by tracking the number of relays in the adjacency list for each node. Moves are accepted or rejected according to the *metropolis criterion* [12], which states that a move is accepted if, and only if, $\Delta f'(S) < 0$ or $random[0,1) < e^{-\Delta f'(S)/T}$ where T is the current temperature and:

$$\Delta f'(S) = \begin{cases} f'(S-u) - f'(S) : \text{Drop only} \\ f'(S+v) - f'(S) : \text{Add only} \\ f'(S-u+v) - f'(S) : \text{Swap} \end{cases} \quad (3)$$

Once a move is accepted, all IDs are invalidated and it is necessary to proceed to the next relay. After this process has been carried out for all relays, a *pass* has been completed and T is reduced by a multiplying factor α, normally in the range [0.85-1). This reduces the possibility of accepting a non-improving solution as the algorithm progresses. The algorithm terminates at the point where no change in $f'(S)$ is seen between the beginning and end of a pass. A minimum number of passes may be specified to avoid accidental, premature convergence and is specified by the algorithm parameter I_{min}. Pseudocode for the MCDS/SA algorithm is given in Fig. 1.

The candidate lists are kept to a constant length in order to keep runtimes down. Initially, nodes are selected from promising two-hop neighbours, with any remaining space populated by randomly chosen, non-relay nodes. On dropping a relay u, one or more nodes in u's adjacency list may become disconnected. It follows that any node capable of reconnecting a newly disconnected node by becoming a relay must reside in the adjacency list of one of the disconnected nodes in question. Therefore, for each node v in the adjacency list of a disconnected neighbour of u, we evaluated $d\,z(v)$ – the number of newly disconnected nodes in v's adjacency list. The nodes v are sorted in descending order of $d\,z(v)$ and placed in the candidate list. Any remaining space in the list is populated randomly in the interests of search diversification. The complexity of this process is $O(\Delta^2)$, where Δ is the maximum degree of any node of the graph. On account of this, it could be argued that the simpler strategy of including all non-relay nodes in the candidate list may be more appropriate for very dense problem instances where Δ approaches n. In practice, however, a huge improvement in runtime was observed using the two-hop candidate list strategy outlined above (see results in section IV).

III. The MCDS/TS Algorithm

The tabu search algorithm for MCDS (*MCDS/TS*) adapts a simple greedy heuristic which we shall call ADD. ADD can be found under different names in a number of papers, including [9] and [10] and is described as follows: Initially all nodes of the graph are marked disconnected. The node with highest degree is added to S and its neighbours are marked dominated. At each subsequent step, the dominated node with the most disconnected neighbours (or highest *yield*) is added to S, and its neighbours are marked dominated. The algorithm terminates when the highest yield reaches zero.

A simple multi-start algorithm might run several passes of ADD, marking all nodes in S tabu after each pass, and repeating ADD with the constraint that only non-tabu nodes may be chosen for the first k steps, where $0 < k < |S|$. While this produces improved results for small k, the process may be enhanced to conduct a more thorough search. The concept is this: A variable depth search (VDS) procedure may remove a set of components c from a solution and replace them with components not in c. The solution may be re-evaluated and accepted or rejected according to some suitable criterion. However, a similar search may be carried out by re-running a greedy algorithm such as ADD and altering one or more component choices at some stage in its pass. This will produce a knock-on effect, the extent of which will vary $|c|$ in a less-controllable manner than for VDS but some control may be achieved from the heuristic principle that $|c|$ is likely to be the larger when alterations occur early in the pass. We note also that solutions may be identified by the alterations made in the behaviour of a greedy algorithm in creating them.

```
procedure MCDS/TS
   GlobalTabuVector ← ŝ ← s ← ∅
   Trees ← 0
   TreePtr ← BuildTree(GlobalTabuVector)
   while(not TreePtr = NULL) do
      Alts(s) ← GetAlts(TreeNode)
      s ← ADD(AltList)                          // updates RCLs
      if (NodeDepth<D_max and f(s)< f(ŝ)+d ) then
         CreateChildren()
      end-if
      if ( f(s)< f(ŝ) ) then
         ŝ ← s
         GlobalTabuVector ← GlobalTabuVector ∪ s
      else if (NodeDepth = 1) then
         GlobalTabuVector ← GlobalTabuVector ∪ s
      end-if
         TreePtr ← NextNode(TreePtr)                // DFS
      if(TreePtr = NULL and Trees < T_max) then
         TreePtr ← BuildTree(GlobalTabuVector)
         Trees ← Trees + 1
      end-if
   end-while
   return ŝ
end-procedure
```

Figure 2. Pseudocode for MCDS/TS

So a candidate MCDS solution S may be subjected to perturbations by adding or removing alterations from its corresponding alteration list *alts*(S), whereby each alteration in *alts*(S) is a pair (i,j) instructing ADD to include node j at step i. However, a constraint must be applied when modifying ADD in this manner. With the exception of the first node, only dominated nodes may be chosen at each step. Otherwise, the dominating set produced may not be connected. As a result, a number of rules are introduced for the maintainance of alteration lists:

On creating perturbations to a CDS, S, by applying changes to *alts*(S):

(i). When a pair (i,j) is added to *alts*(S) and i>1, j must have been dominated at step i in the ADD pass which created S.

(ii). Any pair (i,j) added to *alts*(S) must have i greater than its corresponding value for the last pair in *alts*(S).

(iii). Only the last pair in *alts*(S) may be removed.

Rules ii) and iii) imply that elements of *alts*(*S*) are kept in ascending order of *i*. Rule i) assures that only dominated nodes are added to *alts*(*S*), whilst rules ii) and iii) ensure that alterations already held in the list are not invalidated by changes. The result is a means by which perturbations can be made to a CDS ensuring that the resulting solution is also a valid CDS.

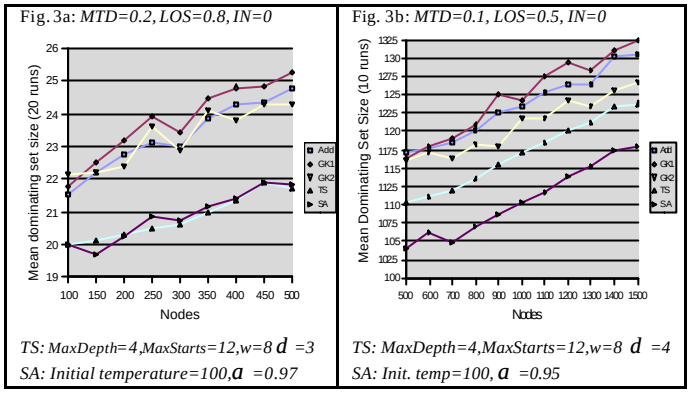

Figure 3. Plots of dominating set size against *n* for constant MTD, IN and LOS

Rule i) is enforced by maintaining a *Restricted Candidate List* (*RCL*) for each step of an ADD pass, recording the yields of the best *RCL_length* candidates at that step. Alterations are chosen from these lists, which guarantees that they will be viable. Rules ii) and iii) are enforced by organising pairs into a tree structure. Every pair must have *i* greater than its parent's and has a corresponding pass whose alteration list is found by reading back up the tree.

Trees are constructed dynamically. Initially, the root node is built with *i*=1 and *j* indexing the highest degree node not marked by the *global tabu vector*. The global tabu vector marks all nodes chosen by root passes and passes producing new global best-known solutions and, as the global tabu vector is initialised as empty, the first tree will select the root node ADD would have selected at step 1. Passes yielding good quality solutions will select pairs from the corresponding RCLs to add *w* children up to a maximum depth D_{max}. This selection process involves scanning RCLs for steps *i*+1 to |*S*| - looking for the highest *w* yields obtained by nodes not used in an ancestor pass. The tree is traversed in depth-first order and when traversal is complete another tree is built up to a limit of T_{max} trees.

A number of rules could potentially be applied to decide which passes will be permitted to bear children. The natural criterion to use is that only passes creating solutions *S* with $|S| < |S_{best}|+d$ may bear children. Here S_{best} is the best known solution so far and d is a parameter of the algorithm, usually an integer in the range $0< d <6$. The extent to which the algorithm dynamically prunes the tree is controlled by d : a lower value for d results in more aggressive pruning. Studies on the effect of altering d are provided for a variety of randomly-generated problem instances in section IV. Pseudocode for MCDS/TS is given in Fig. 2.

By comparison to conventional TS approaches. the root created in building a new tree may be considered equivalent to a multi-start, while passes further down the tree create a progressively more localised search akin to strategic oscillation. However, knock-on effects ought to produce considerable variation in the size and extent of the perturbations applied. If necessary, elite candidates may be referred to by their alteration list and reconstructed using ADD if a further search is desired around them.

IV. RESULTS

A. Comparison of CDS Sizes

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.

In the absence of problem instances for which the optimum solution is known (other than small instances that can be optimised completely through exhaustive search), it was necessary to find some suitable benchmark against which to compare performance. As a result, three greedy heuristics were used for comparison. These were the ADD algorithm [9] and Guha and Khuller's two algorithms [10], hereafter referred to as GK1 and GK2. It is expected that these three algorithms would be outperformed by the metaheuristics.

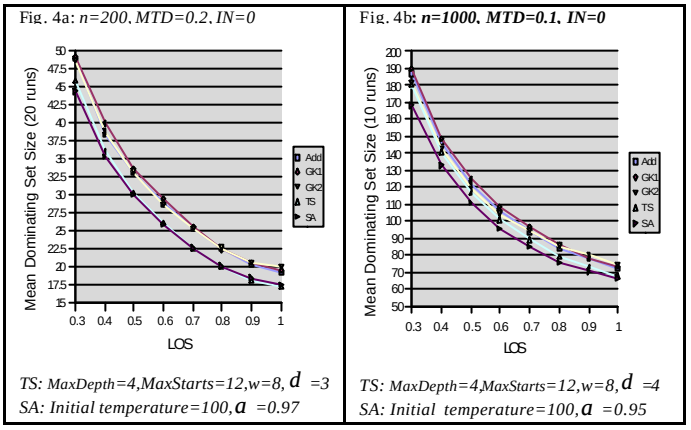

Figure 4. of dominating set size against LOS for constant MTD, IN and *n*

Initially, the effect of varying node count (*n*) upon dominating set size (|*S*|) was observed for constant values of the other problem parameters MTD, LOS and IN. Following this, the effect of increasing edge density upon |*S*| was studied by varying LOS and keeping the remaining parameters constant. Finally, the algorithms' abilities to handle node constraints was investigated by increasing IN and observing the resulting values for |*S*|. For MCDS/SA, the candidate list lengths were set to 20, P_{move} was set to 0.1 and I_{min} to 40. For MCDS/TS, the *RCL_Length* parameter was set to 4.

Fig. 3 shows the effect of increasing *n* on |*S*| for small, dense problem instances (3a) and large, sparse ones (3b). Whilst there is little difference between SA and TS for smaller graphs (3a), SA outperforms TS on larger ones (3b). Furthermore, comparing 3a and 3b for n=500, shows TS

giving better solutions for MTD = 0.2 and LOS = 0.8, while SA is far more successful with MTD = 0.1 and LOS = 0.5. This shows that edge density also has a bearing on the two algorithms' relative performance.

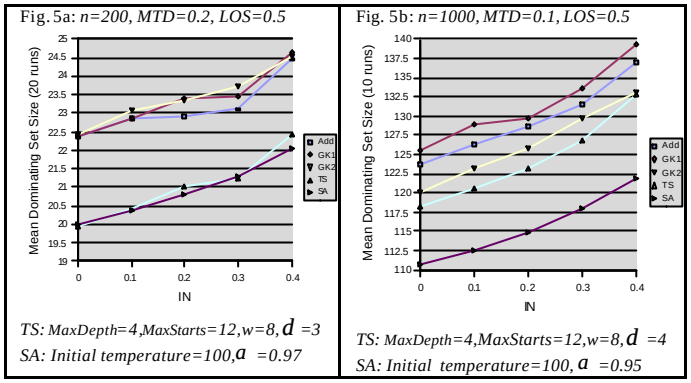

Figure 5. Plots of Dominating set size against IN for constant *n*, MTD & LOS

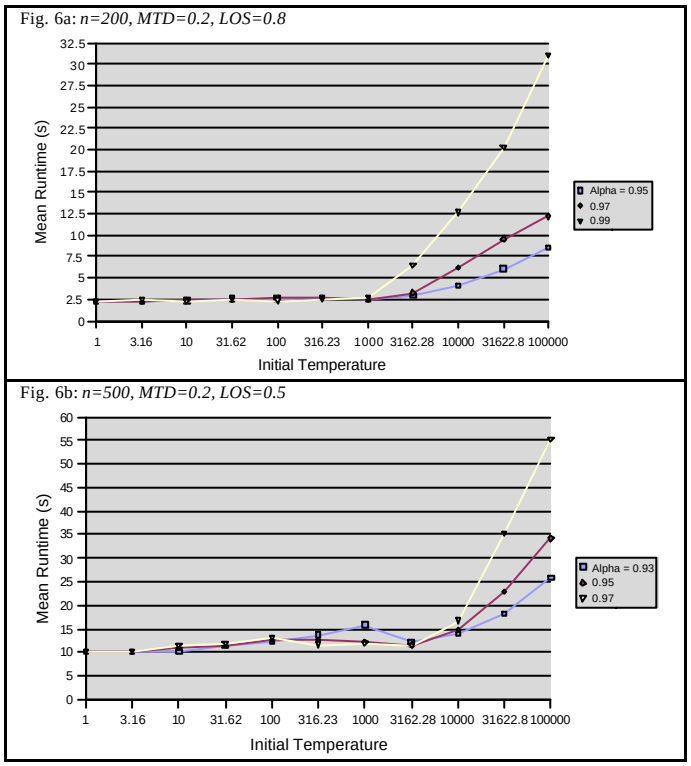

Figure 6. Time Charts for MCDS/SA

Fig. 4 confirms this, showing the result of varying LOS for constant *n* and MTD. In 4a, TS begins to outperform SA as the density increases and, although SA is superior in 4b, a relative improvement can be observed for TS. The effect of increasing edge density explains the inconcistency in results for Fig. 3. Comparing 3a and 3b, 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* (for constant MTD and LOS).

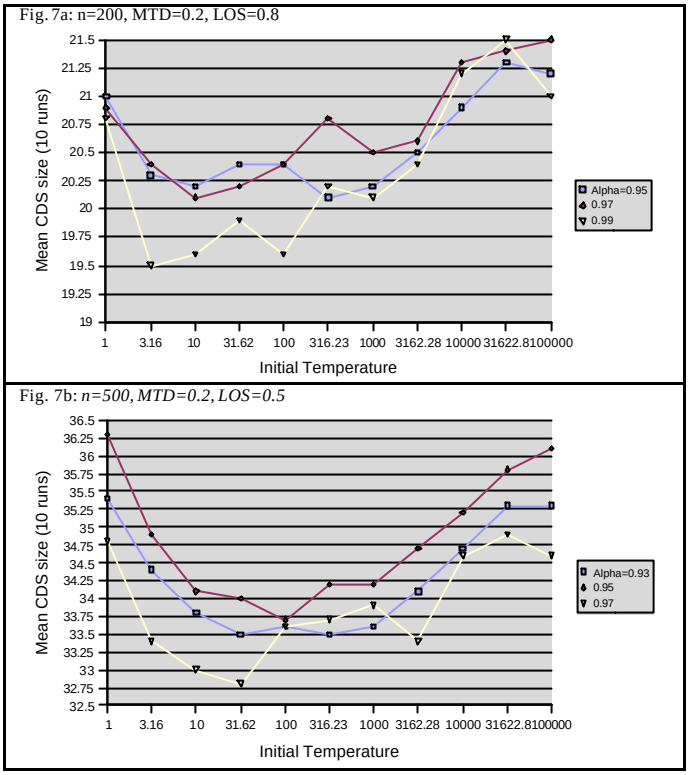

Figure 7. Corresponding CDS Sizes for MCDS/SA Time Charts

Whilst Figures 3 and 4 deal exclusively with edge constraints (IN=0), the effect of increasing IN is observed in Fig. 5. 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.

B. Runtimes and Parameter Settings

The complexity of MCDS/SA using constant length candidate lists is $O(n(\Delta^2 + m))$ or $O(n^3)$ per pass, although in practice performance is normally better than this. Runtimes are given in Fig. 6 with various values for the initial temperature and α. Here it can be seen that runtimes do not begin to increase until the initial temperature approaches 1000. Fig. 7 shows the corresponding CDS sizes for the times in Fig. 6. Here we can see, fortunately that lower values for initial temperature tend to give better results as well. A good compromise between runtime and solution quality would be to set the initial temperature between 3 and 100, perhaps slightly greater for instances with larger n. Where α is concerned it appears that high values work better without compromising runtimes too badly. In tests, runtimes were generally around 2-3 seconds for n=200, 10-15 seconds for n=500 and 50-70 seconds for n=1000, using an Athlon 1700XP processor and 512MB RAM, with algorithms implemented in non-optimised C++.

MCDS/SA was tested with three different graph types, ***A*** (n=200, MTD=0.2, LOS=0.8, IN=0), ***B*** (n=500, MTD=0.2, LOS=0.5, IN=0) & ***C*** (n=1000, MTD=0.1, LOS=0.5, IN=0). Runtime increases and CDS size decreases with increasing I_{min} so the setting of this parameter is largely a trade-off between

runtime and solution quality. There is also a suggestion that a larger I_{min} may be more effective with smaller instances. A positive P_{move} is certainly advantageous, with a range of values between 0.1 and 0.4 being potentially useful. The effect on runtimes is less predictable, but it appears that none of the tested settings for P_{move} are overly-detrimental. Runtimes increase consistently with increasing candidate list length and little improvement in solution quality is found after the list length exceeds 20. This indicates that the optimum trade-off length for candidate lists may be found somewhere in the region of 20 for a variety of graph dimensions.

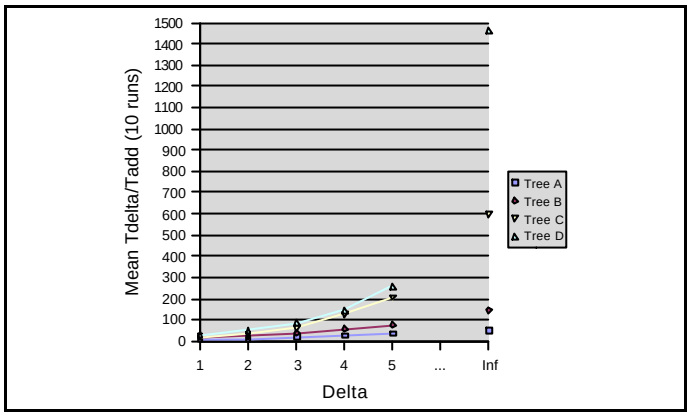

Figure 8. Time Charts for MCDS/TS

Runtimes for MCDS/TS are controlled primarily by d. It should be fairly obvious that MCDS/TS complexity is exponential with respect to D_{max} and that the purpose of d is minimise the potentially catastrophic impact this could have on runtimes. Further precautions include setting D_{max} to a small constant and increasing w and T_{max} to improve the search. The complexity of an ADD pass is $O(n^2)$ in this implementation, although other papers [10] claim bounds as low as $O(m)$. Fig. 8 shows the mean ratio T_{delta}/T_{add} for increasing d, where T_{delta} is the time taken for MCDS/TS to compute for the given value of d and T_{add} is the time taken for one ADD pass. Fig. 9 shows the corresponding percentage reduction in CDS size for MCDS/TS when compared to ADD. It can be seen from Fig. 9 that relatively little improvement in CDS size can be gained by increasing d above 4 or 5. However, Fig. 8 shows that runtimes are particularly small when compared to the unrestricted case where $d = \infty$. As a result, values for d around 3-5 with D_{max} no greater than 5 would seem to give a good compromise between runtime and solution quality. Runtimes were of a similar order to MCDS/SA, ranging roughly from 1 second to one minute (for the same hardware/software specification as before), depending on the size of the problem graph.

V. CONCLUSIONS

It can be seen from the results that each of the algorithms has its preferred variety of problem instance. SA seems to prefer large, sparse problem instances, whereas TS prefers small, dense ones and a number of factors may be involved in attempting to explain this behaviour. Firstly, on considering the relative performance of ADD when compared to GK2, we see that ADD (the algorithm on which MCDS/TS is based) improves its performance when compared to GK2 as edge density increases, but that GK2 comes into its own as the graphs get larger or less dense. The reason may be conjectured as follows: ADD requires the dominating set to be connected at every step, while GK2 builds a dominating set and connects it up later. As a result, ADD possesses the ability to ignore high-degree nodes early on in the search whilst GK2 does not. The extent to which this adversely affects ADD's performance ought to increase as the graph becomes larger or more sparse. As MCDS/TS is based on ADD, it is little surprise that it exhibits similar behaviour when compared to MCDS/SA.

A second factor may help to explain the relative performance of these two algorithms. Whilst the d-pruning of MCDS/TS trees goes some way to reducing runtime, it is still difficult to increase D_{max} enough to produce a really thorough search without the runtimes becoming too high. Naturally, in large sparse networks, the CDSs are likely to be larger, so many alterations will occur early in the search. As a result the search is all diversification and very little intensification. This is likely to give inferior results as good regions of the search space may be left unexploited. It may be that a different approach to pruning could provide improvements for the MCDS/TS algorithm.

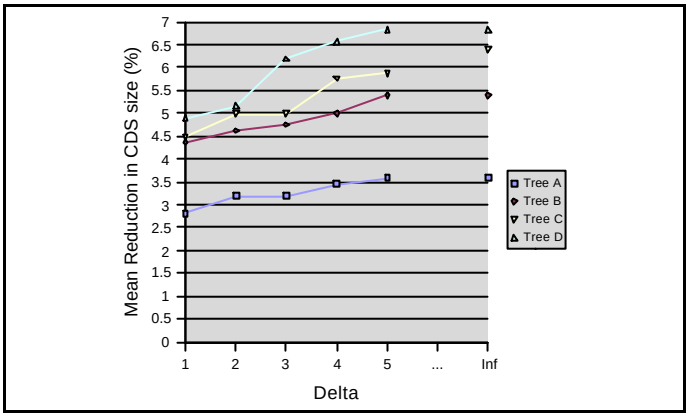

Figure 9. Corresponding Reduction in CDS size for MCDS/TS Time Charts

These observations lead to a number of insights into the development of heuristics for MCDS. Firstly, the possible use of hyper-heuristics may be considered, whereby the size and density of the problem is measured and an algorithm selected accordingly. As discussed, this does not only apply to the two metaheuristics, but also to simple algorithms like ADD and GK2 which are both useful methods if time is an issue. Secondly, the development of hybrid techniques might be considered. Possibilities include using zero temperature annealing on elite candidates from MCDS/TS or employing MCDS/TS with a shallower tree to generate a starting point for the annealing process.

In addition to the above proposals for future work, there is still the need to address further constraints. Minimum k-

connected dominating sets (where k>1) and capacitative minimum connected dominating sets might be considered, for example. The handling of redundancy and load constraints will introduce yet another level of complexity into an already tough search problem. It is not yet known whether the kind of methods outlined here could be adapted, or if further techniques would need to be developed in order to achieve this.

REFERENCES

[1] E. Aarts and J. Korst, " Simulated Annealing and Boltzmann Machines", Wiley, 1989.

[2] K. Alzoubi, P.J. Wan and O. Frieder, "Distributed Construction of Connected Dominating Set" in Wireless Ad Hoc Networks, Proceedings of IEEE INFOCOM, Vol. 3, pp1597-1604, June 2002.

[3] S. Butenko, X. Cheng, C. Oliveira and P. Pardalos, "A new algorithm for connected dominating sets on ad hoc network"s, in: Butenko, S. Murphey, R. and Pardalos, P., Recent Developments in Cooperative Control and Optimization, pp61-73. Kluwer, 2003.

[4] R. Cahn, "Wide Area Network Design", Morgan Kauffman, 1998.

[5] F. Dai and J. Wu, "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, pp908-920, October 2004.

[6] M. Garey and D.S. Johnson, "Computers and Intractability, a Guide to the Theory of NP-Completeness", Freeman, 1979.

[7] F. Glover and M. Laguna, "Tabu Search", Kluwer Academic Publishers, 1997.

[8] F. Glover, "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. pp1-24. Kluwer, 2000.

[9] V. Grout, "Principles of Cost Minimisation in Wireless Networks", Journal of Heuristics, Vol. 11: pp115-133, 2005.

[10] S. Guha and S. Khuller, "Approximation Algorithms for Connected Dominating Sets", Algorithmica, 20, pp374-387, 1998.

[11] M. Kouider and P. Vestergaard, "Generalized Connected Domination in Graphs", Discrete Mathematics and Theoretical Computer Science, 8, pp57-64., 2006.

[12] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller and E. Teller, "Equation of State Calculations by Fast Computing Machines", Journal of Chemical Physics, 21, pp1087-1092, 1953.

[13] J. Wu, W. Lou and F. Dai, F., "Extended Multipoint Relays to Determine Connected Dominating Sets in MANETs", IEEE Transactions on Computers, Vol. 55, No. 3, pp334-347, March 2006.