A New Algorithm for the Euclidean k-Bottleneck Steiner Problem

Marcus Brazil, Charl Ras, and Doreen Thomas

***Abstract*— We consider the problem of adding a fixed number of relays to a WSN in order to minimise the length of the longest transmission edge. Since routing subnetworks are often trees we model the problem as a Euclidean k-bottleneck Steiner tree problem (k-BSTP). We then propose a new iterative approximation algorithm for the k-BSTP, based on an exact solution to the 1-BSTP, and compare our heuristic (via simulation) to the currently best performing heuristic in the literature, namely the minimum spanning tree heuristic (MSTH). We observe that our algorithm performs up to 8% better than MSTH on uniformly distributed node-sets.**

I. Introduction

Network problems where the longest (or most costly) edge is to be minimised by introducing new points are commonly referred to as *bottleneck Steiner problems* (a similar problem in location science is the *multifacility minimax location problem*). Of the many applications where these models are suitable, we are most interested in finding lifetime optimising *relay augmentation* strategies for wireless sensor networks (WSNs), i.e., strategies based on the deployment of additional non-sensing transmitters. The utility of these strategies lie in their ability to deterministically alter the topology of a network, thereby providing the routing protocol with spanning subnetworks satisfying the relevant energy-saving objectives. Relay augmentation is also a powerful and relatively inexpensive method of optimising many other topology-dependent WSN objectives; see for instance [5], [16], [9], [13].

In this paper we make the justified assumption ([8], [15], [17]) that a routing tree will be constructed by the WSN during network initialisation. In addition we assume that every sensor and relay transmits data at the same rate. This second assumption is valid for many types of WSN applications, especially those where data does not accumulate. Examples include any type of average measurement, such as temperature or humidity; measurements of maximums or minimums of some environmental property; and intrusion detection or other early-warning systems. We define (as most authors have done previously) the lifetime of a WSN as the time until first node failure due to battery depletion. In most WSNs the bulk of a sensor's energy consumption is attributable to data transmission. In fact, energy consumption is generally proportional to traffic flow multiplied by some power (usually between 2 and 4) of the transmission distance. This fact, together with our assumptions, implies that an inverse relationship exists between the lifetime of a network and the maximum communication range, $R_{\max}$, over all pairs of sensors. In this paper we also assume that a desired level of coverage has been attained by some initial (random or deterministic) network deployment phase, and that a *fixed* number of relays are available for a deterministic lifetime-extending (i.e., $R_{\max}$ reducing) augmentation phase.

We model our relay augmentation strategy as the Euclidean *k-bottleneck Steiner tree problem* (k-BSTP). Phase-one sensors are represented by fixed points (*terminals*) embedded in the Euclidean plane, and phase-two relays are the variable nodes (*Steiner points*) of the network. The objective is then to minimise the longest edge of a tree interconnecting all nodes, under the constraint that at most k Steiner points may be introduced.

The bottleneck Steiner problem was introduced by Saraffzedeh and Wong [11], and various authors have considered the problem in the context of facility location science [4], [10]. The k-bottleneck Steiner tree problem is NP-hard, in fact, no polynomial-time algorithm exists for the problem in the Euclidean plane with a performance ratio less than $\sqrt{2}$ unless P=NP. Wang and Du, the authors who demonstrated this complexity bound, present in [12] the first deterministic approximation algorithm for the k-bottleneck problem in both the rectilinear and Euclidean planes. They also prove that the performance ratio of their algorithm, the so-called *minimum spanning tree heuristic* (MSTH), is bounded above by 2 in both these planes. Du et al. [6] describe a randomised heuristic algorithm with a performance ratio of $\sqrt{3}+\epsilon$.

In this paper, which essentially summarises the most important findings of [3], we introduce a new heuristic for the k-bottleneck Steiner tree problem as applied to WSNs. We present simulation results that demonstrate a significant improvement in performance of our heuristic over the currently best performing deterministic approximation algorithm for the problem. Section II formalises our network model. In Section III we provide the details of our *iterative* 1-*BST heuristic* approach, and provide an analysis, with examples, of the algorithm in Section IV. Section V is devoted to a substantial empirical demonstration of the performance of our algorithm. We summarise our conclusions and sketch an outline of our future research objectives in Section VI.

II. Sensor Network Model and Properties

We assume that all sensors and relays have the same maximum communication range R, and that they are able to adjust their transmission power at network start-up to any

This work was funded by an Australian Research Council Discovery-grant

Marcus Brazil and Charl Ras are with the Department of Electrical and Electronic Engineering, University of Melbourne, Parkville, VIC, 3010, Australia brazil@unimelb.edu.au

Doreen Thomas is with the Department of Mechanical Engineering, University of Melbourne

ISBN 978-963-311-370-7

value below R. The energy consumption per packet is given, as in [13], by

$$\mathbb{E}(r) = r^{\alpha} + c$$

where r is packet transmission distance, c is a constant and is small relative to r^{α}, and $\alpha \in [2, 4]$. Power consumption attributable to packet receipt is assumed to be negligible.

Throughout this paper $|\cdot|$ denotes the Euclidean norm. We define the disk graph $D_R = \langle X, E\rangle$ of disk radius R on a set of terminals X by setting $E = \{\langle x, y\rangle : x, y \in X, |xy| \leq R\}$. A disk graph on all data-source points and the single base station, contains (under the assumption that D_R is connected) all possible routing trees for the WSN. We assume that a single routing tree, which will in fact be a minimum spanning tree (MST), is selected at network start-up for all subsequent (until first node-death) communication. Given k additional relays, we need to complete the task of positioning these relays in order to maximise the lifetime of an optimal routing tree.

Let $\ell_{\max}(T)$ denote the length of the longest edge in a tree T. For any set X of embedded terminals, let $\mathcal{C}(X, k)$ be the set of all trees interconnecting X and at most k other points in the plane.

Definition. The *k-bottleneck Steiner tree problem*, or k-BST problem, has the following form:

Given A set X of n points (terminals) in $\mathbb{R}^2$, and a positive integer k.

Find A set S (the Steiner points) of at most k points in $\mathbb{R}^2$, and a spanning tree T on $X \cup S$ such that $\ell_{\max}(T) \leq \ell_{\max}(T')$ for any $T' \in \mathcal{C}(X, k)$.

We refer to T as a *k-bottleneck Steiner minimum tree* (k-BSMT). In general, k-BSMTs are not easy to construct. Currently, the most general (with regards to the underlying metric) exact algorithm for calculating k-BSMTs has a complexity of $\mathcal{O}(g(k) \cdot n^{2k})$, where $g(k)$ is exponential (see [2]). When k is large or is, say, an increasing function of n, then this exact algorithm must be replaced by a good heuristic. The following important lemma is derived from [2].

Lemma 1: For any set of terminals there exists a k-BSMT that is an MST on its complete set of nodes and such that the maximum degree of any node is 5.

Minimum spanning trees play a key role in our main algorithm. As described in [2], by utilising a preprocessing stage requiring $\mathcal{O}(n^2)$ time one can update a given MST within constant time to include a new point. The preprocessing subroutine, which we denote by PREPROCESS, takes as input an MST T and then, for every pair of nodes of T, calculates the longest edge on the path connecting the nodes. The results of this calculation are stored in a table H, which can then be used to "read off" the edges that need to be deleted from T after a new point, together with new edges to its neighbours in T, is introduced.

By considering the 1-BST problem on three terminals one may also grasp the importance of minimum-radius covering circles to the general k-BST problem. The *Chebyshev centre* of a set of points is the centre of the smallest circle covering the points, and is always determined by two or three points lying on the circumference of the covering circle.

The results of this section now provide us with a simple (albeit relatively time expensive) exact solution to the 1-BST problem: we select the cheapest tree after iterating through all subsets of at most five neighbours for the Steiner point where, for every subset, the Steiner point is located at the Chebyshev centre of its neighbours. A preconstructed MST on the given terminals is then updated in constant time to include the new Steiner point. The total complexity of this algorithm is therefore $\mathcal{O}(n^5)$.

III. Algorithms

Exact solutions to the 1-BST problem in the Euclidean plane have recently been proposed by Brazil et al. [2] and Bae et al. [1], utilising *oriented Dirichlet cell partitions* for an $\mathcal{O}(n^2)$ solution and *farthest colour Voronoi diagrams* for an $\mathcal{O}(n \log n)$ solution respectively. We provide a brief description of the exact solution found by Brazil et al. for constructing k-BSMTs.

A. An Exact Solution to the k-BSTP

To construct a k-BSMT the authors in [2] first partition the plane into an *overlayed oriented Dirichlet cell partition*. For a given set of terminals X, and a minimum spanning tree T on X, this partition, $\mathcal{Q}$, has the following properties.

1) Each region W of $\mathcal{Q}$ is generated by at most six terminals $C(W) = \{\mathbf{t}_1, ..., \mathbf{t}_6\}$,
2) If a new point $\mathbf{s}$ is embedded anywhere in W, then there exists an MST T' on $X \cup \{\mathbf{s}\}$ such that $E(T') \subset \{\mathbf{st}_i\} \cup E(T)$, where $E(G)$ represents the edge-set of a graph G.

Once $\mathcal{Q}$ has been constructed (which takes $\mathcal{O}(n^2)$ time) the algorithm iterates through all choices $\{W_1, ..., W_k\}$ of up to k regions of $\mathcal{Q}$ (with repetition), introducing a Steiner point $\mathbf{s}_i$ into region W_i. For each set of Steiner points $S = \{\mathbf{s}_i\}$ a graph G_S is constructed with node-set $S \cup (\bigcup C(W_i))$ and edge-set $(S \times S) \cup \{\mathbf{s}_i\mathbf{t}_j : \mathbf{s}_i \in S, \mathbf{t}_j \in C(W_i)\}$.

One by one, every subtree F of G_S of maximum degree at most five is then added to T, and the resultant graph is updated into an MST using a method that generalises the single-point update routine mentioned in the previous section. The exact positions of the $\mathbf{s}_i$ (in their respective regions) are calculated using a fully polynomial-time approximation scheme, but for $k = 1$ this simply reduces to placing the Steiner point at the Chebyshev centre of its neighbours. The total complexity of the algorithm, excluding factors not containing n, turns out to be $\mathcal{O}(n^{2k})$.

The construction of $\mathcal{Q}$ at the start of the algorithm proceeds as follows. For any two directions ϕ_i and ϕ_j in the plane, we denote by $K(\mathbf{y}, \phi_i, \phi_j)$ the acute cone with vertex $\mathbf{y}$ and limiting rays specified by ϕ_i and ϕ_j. For each $i = 0, ..., 5$, the ith *oriented Dirichlet cell* (ODC) of $\mathbf{w} \in X$ is the set:

$$\{\mathbf{y} \in \mathbb{R}^2 : ||\mathbf{w}-\mathbf{y}|| = \min\{||\mathbf{x}-\mathbf{y}||, \mathbf{x} \in X \cap K(\mathbf{y}, \theta_i, \theta_{i+1})\}\}$$

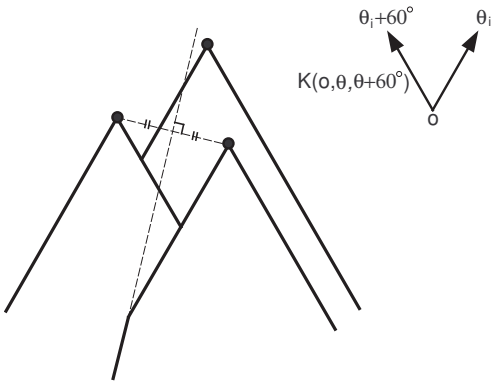


Fig. 1. Example of an ODC partition

where $\theta_0 = 0^\circ$ and $\theta_{i+1} = \theta_i + 60^\circ$. In other words, this is the set of all points $\{\mathbf{y}\}$ whose closest terminal in the cone $K(\mathbf{y}, \theta_i, \theta_{i+1})$ is $\mathbf{w}$. The set of ith ODCs on X, called the *ith ODC partition of X*, is a type of Voronoi diagram, and is relatively easy to construct (see [2] or [7], and see Fig. 1 for an example of an ODC partition on three terminals). Overlaying (i.e. finding the coarsest common refinement) all six ith ODC partitions results in the required partition $\mathcal{Q}$.

Neither of the exact 1-BST algorithms of [1] or [2] have been implemented in practice, and it seems that this would be a relatively complex task given the sophisticated geometrical nature of these solutions. Moreover, the complexity formulas of these algorithms appear to contain large hidden constants. We therefore use the $\mathcal{O}(n^5)$ procedure of the previous section for our main algorithm, which we augment with a pruning subroutine that considerably reduces the average running time in practice.

B. The Minimum Spanning Tree Heuristic

The only deterministic algorithm in the literature for approximating a k-BSMT is the *minimum spanning tree heuristic* (MSTH) of Wang and Du [12]. Let T be an MST on any q nodes. To *bead* an edge e_i of T with n_i degree-two Steiner points (*beads*) we place n_i equally spaced beads along e_i. For each e_i in T let $l(e_i) = \dfrac{|e_i|}{n_i + 1}$.

Algorithm 1 Subroutine BEAD (MSTH)

Input: An MST T on q nodes and a positive integer j
Output: The beaded MST T'

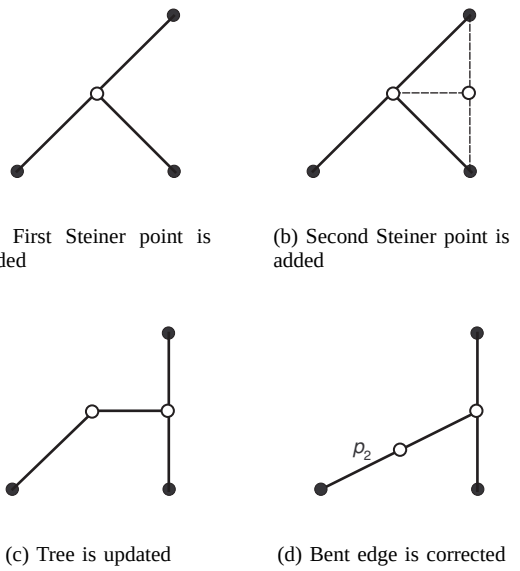
1: Clear all beads of T, i.e., set $n_i = 0$ for any i
2: Let $e_1, ..., e_{q-1}$ be the edges of T
3: Compute $l(e_i)$ for each e_i
4: Sort the edges in non-decreasing order of $l(\cdot)$
5: Add a bead to e_i of largest $l(\cdot)$ value
6: Update $l(e_i)$
7: Relocate the beads on e_i so that they are equally spaced
8: Reset e_i's position in the ordering
9: Repeat Steps 5-8 until j beads have been added

Our main algorithm makes extensive use of MSTH as a "look-ahead" component, which we denote by BEAD.

C. A Naive Iterative 1-Bottleneck Steiner Tree Heuristic

A naive *iterative 1-BST heuristic* (or I1-BSTH) on a set X of terminals might proceed as follows. First we construct an MST T on X and then execute PREPROCESS in order that T may be updated in constant time to include a new point. An optimal 1-BST on X is then constructed by iterating through all possible sets of neighbours for the Steiner point, embedding the Steiner point at the Chebyshev centre of its neighbours for every iteration, and selecting the cheapest tree. This sequence of routines is repeated $k-1$ times, i.e., until k new points have been added. In order to compare the performance of this heuristic against MSTH we implemented both in C (with minor modifications as described below in Subroutine CONVERT), and tested them on uniformly distributed terminal sets. Roughly speaking, simulations show a maximum average improvement of approximately 4% for naive I1-BSTH over MSTH. This average can be increased to about 5% by combining naive I1-BSTH and MSTH into a meta-heuristic (i.e., a heuristic that picks the best of the two constructions), which shows that MSTH can outperform naive I1-BSTH on some terminal sets. A simple example where this occurs is described next, and is illustrated in Figs. 3 and 2. In these figures (and throughout) the solid circles are terminals and the open circles are Steiner points.

Example 1: Let $k = 2$ and consider three terminals on the vertices of a unit square. Then $p_1 = 0.5$ and $p_2 = \sqrt{5}/4$ represent the longest edges in the solutions of MSTH and naive I1-BSTH respectively, with $p_1 < p_2$ (see Figs 3 and 2).



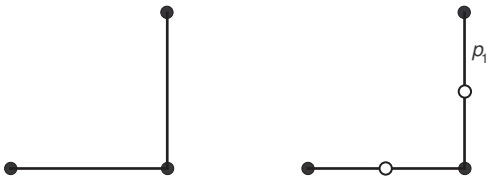
(a) First Steiner point is added

(b) Second Steiner point is added

(c) Tree is updated

(d) Bent edge is corrected

Fig. 2. The solution constructed by naive I1-BSTH as described in Example 1



(a) Three terminals on a square plus their MST

(b) Longest edges are beaded

Fig. 3. The solution constructed by MSTH as described in Example 1

Fig. 2 also illustrates one of the modifications made to naive I1-BSTH, namely that of correcting any *bent edges* that may appear in the final solution; a bent edge actually consists of two edges incident, at an angle of less than $180°$, to a Steiner point of degree two. Bent edges can clearly be straightened without increasing the length of the longest edge in a tree.

D. The Pre-beaded Iterative 1*-Bottleneck Steiner Tree Heuristic*

Fig. 2 highlights a major draw-back of iterative greedy heuristics: an optimal solution at some iteration may undercut the gains of subsequent iterations. We notice that, bent-edge correction aside, the tree from Fig. 2(c) provides no decrease in maximum edge-length over Fig. 2(a). This shortcoming permeates the class of solutions constructed by naive I1-BSTH. A look-ahead routine is therefore warranted as a modification to 1I-BSTH, but only if gains in performance are made by such a routine over the afore-mentioned meta-heuristic.

The *Pre-beaded iterative* 1*-BST heuristic* (Pre-beaded I1-BSTH), described in Algorithm 2, utilises subroutine BEAD as a look-ahead component. Pre-beaded I1-BSTH also contains the procedure CONVERT, which consists exactly of the modifications that were made to naive I1-BSTH for its implementation in C, and the function FEASIBLE (see below) for reducing the number of neighbour-sets that must be considered. We also include a simple procedure for pruning away some infeasible Steiner point neighbour-sets, which reduces average run-time by approximately 50% for any n, k pair. The function FEASIBLE in Line 7 of Algorithm 2 refers to this pruning technique and is dependent on the following lemma.

Lemma 2: Let T be an MST on X, and suppose that T is updated with a Steiner point $\mathbf{s}$ such that the resulting tree T' satisfies $\ell_{\max}(T') \leq \ell_{\max}(T)$. Then $|\mathbf{xx}'| \leq 2 \cdot \ell_{\max}(T)$ for any neighbours $\mathbf{x}, \mathbf{x}'$ of $\mathbf{s}$ in T'.

Proof: $\frac{1}{2}|\mathbf{xx}'| \leq \max\{|\mathbf{sx}|, |\mathbf{sx}'|\} \leq \ell_{\max}(T') \leq \ell_{\max}(T)$, where the first inequality follows from the triangle inequality. ■

Algorithm 2 Pre-beaded I1-BSTH

Input: A set X of n points in the plane, and a positive integer k

Output: An approximate k-BSMT T interconnecting X and k new points

1: Construct a minimum spanning tree T_0 on X
2: Let $p = c = 0$
3: **while** $c < k$ **and** $p < k$ **do**
4: Run BEAD with input T_c and $k - 1 - p$, and output T_c'
5: Run PREPROCESS with input T_c' and output H
6: Order the y-coordinates of the nodes of T_c' and store the sequence in L_y
7: **for** every FEASIBLE neighbour-set $X' \subseteq X$ **do**
8: Let $T' = T_c'$ and let $\mathbf{s}$ be the new Steiner point of X'
9: Delete the edges of T' specified by X' and H
10: Run CONVERT with input T' and $\mathbf{s}$, and output T''
11: Let p' be the number of non-bead Steiner points of T''
12: Run BEAD with input T'' and $k - p'$, and output T^*
13: **end for**
14: Increment c by 1, let T_c be the cheapest T^* produced, and let p be the number of non-bead Steiner points of T_c
15: **end while**

Algorithm 3 Subroutine CONVERT

Input: A tree T interconnecting a set of terminals X and a set of Steiner points S, and an element $\mathbf{s} \in S$

Output: The modified tree T'

1: Delete all degree-one Steiner points of T
2: Correct all bent edges and relocate beads to be equally spaced along edges
3: Relocate $\mathbf{s}$ at the Chebyshev point of its neighbours, and add the edges incident to $\mathbf{s}$

The construction of feasible neighbour-sets occurs during each main iteration. At the start of an iteration (Line 6) the algorithm orders the complete set of nodes by their y-coordinates, and stores the sequence in L_y. Each FEASIBLE neighbour-set of up to at most five nodes is then built using Subroutine FIND-NEIGHBOUR. The real number ℓ in the input of the subroutine is equal to $\ell_{\max}(T_c')$ and is calculated (without extra complexity cost) during the BEAD subroutine of Line 4.

Algorithm 4 Subroutine FIND-NEIGHBOUR

Input: A tree T, a subset X' of at most four nodes of T, an element $\mathbf{x} \in X'$, a positive real number ℓ, and a y-coordinate ordering L_y of the nodes of T

Output: A node $\mathbf{y}$ of T such that $\mathbf{y} \notin X'$ and $|\mathbf{yx}'| \leq 2 \cdot \ell_{\max}(T)$ for every $\mathbf{x}' \in X'$

1: Scan consecutively to the left and right of $\mathbf{x}$ in L_y
2: Stop when a point $\mathbf{y}$ is reached that does not belong to X' and is within distance 2ℓ of every node in X'

Lastly, observe that Line 9 encompasses the MST update routine discussed in Section II, where the correct edges to delete are specified by H.

IV. Analysis of Pre-beaded I1-BSTH

There are three aspects of Pre-beaded I1-BSTH that we will examine more closely in this section. The first is the look-ahead component actualised by the call to BEAD in Line 12 of Algorithm 2. Calculating an optimal sequence of Steiner point additions can generally not be done greedily, and calculating every potential sequence would take us into exponential complexity. However, we can remain within polynomial complexity if we assume, at any given iteration of the algorithm, that the remaining sequence of additions will consist of beads only. This assumption can be viewed as providing a lower bound for the overall (including past and future iterations) performance of an optimal Steiner point addition sequence. This lower bound is clearly equal to the performance of MSTH, and hence we are guaranteed that Pre-beaded I1-BSTH never performs worse than MSTH. This result on its own is not satisfactory, since the meta heuristic combining the naive version of I1-BSTH and MSTH has the same property. But as will be demonstrated in the next section, algorithm Pre-beaded I1-BSTH performs significantly better than the meta heuristic.

The second aspect we would like to examine concerns the call to BEAD in Line 4. This call ensures that there are $k-1$ Steiner points in the input tree before any main iteration. Consequently our algorithm may be viewed as beading the entire initial MST, and then one-by-one removing beads and adding them back optimally. It is possible to construct a version of I1-BSTH that does not utilise this BEAD call but still contains the look-ahead routine. This so called "post-beaded" version performs very similarly to our pre-beaded algorithm, but falls slightly short in the long run.

Finally, we look at complexity. Since the time required by BEAD is $\mathcal{O}(n \log n)$, the theoretical complexity of I1-BSTH is $\mathcal{O}(n^6 \log n)$. As mentioned before, however, the pruning routine we incorporate in the function FEASIBLE reduces the practical running-time. This fact has a certain analogy with the recent success of GEOSTEINER (see [14]) in calculating exact Steiner minimal trees. Even though calculating such trees is NP-hard, GEOSTEINER can tractably construct optimal solutions for instances of up to 10 000 terminals.

V. Empirical Results

Over 130 000 k-BST problem instances, uniformly distributed on a 10 000 by 10 000 point grid, were generated and solved by both MSTH and Pre-beaded I1-BSTH on a Pentium 4, 3GHz CPU with 1GB of RAM. The performance of Pre-beaded I1-BSTH was calculated as $P = 1 - \frac{\ell_{\max}(T_B)}{\ell_{\max}(T_M)}$, where T_M and T_B are the solutions generated by MSTH and Pre-beaded I1-BSTH respectively. The number of instances simulated, denoted by N, mostly varied arbitrarily in the range $600 \leq N \leq 2000$, but generally decreased as n and k increased. We focussed on terminal sets of size $n = 10, 20, 30, 50$ and $k = 1, .., n$ and $k = 1.5n, 2n$, and also the pairs $n = 10, k = 50$; $n = 30, k = 75$; $n = 40, k = 11$; $n = 100, k = 33$; and $n = 200, k = 68$. Fig. 4 presents the simulation results for average performance (AP); CPU times lie within the given approximate intervals and are measured in seconds. In Table I we provide the same details for the (apparently) best performing cases for each n; here we also include maximum performance (MP) and standard deviation (SD) results. Fig. 5 details the relative frequency results for these best performing cases, where the notation $(x]$ is used to denote the interval $(x - 0.01, x]$.

A few observations are in order. The first thing to notice about the graph in Fig. 4 is that, for all depicted values of n, performance increases rapidly from $k = 0$, reaching a peak in the approximate range $0.2 \leq \frac{k}{n} \leq 0.34$, whereafter performance decreases at a decreasing rate. In all cases we see near-asymptotic behaviour as k increases beyond $k = n$, apparently approaching a limit near $P = 0.015$.

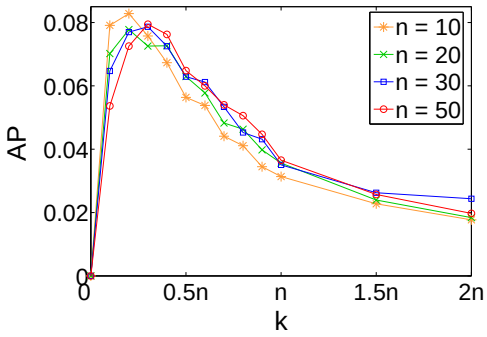
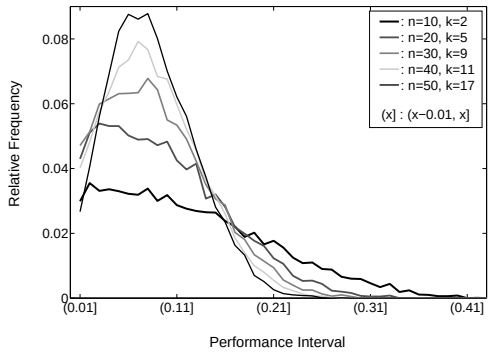


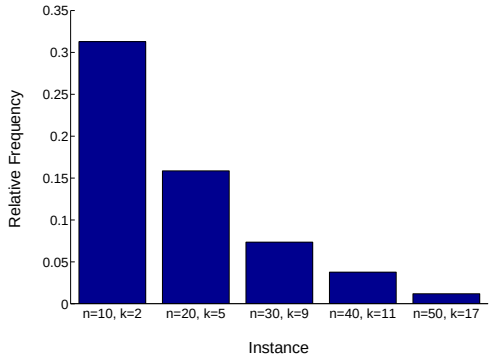
Fig. 4. Simulation results for average performance (AP); 1 sec $\leq$ CPU time $\leq$ 90 sec

TABLE I
Best performing cases

Instance	N	AP	MP	SD
$n = 10, k = 2$	10000	0.0828	0.4140	0.0886
$n = 20, k = 5$	10000	0.0785	0.3720	0.0668
$n = 30, k = 9$	10000	0.0789	0.3050	0.0564
$n = 40, k = 11$	10000	0.0789	0.3520	0.0498
$n = 50, k = 17$	10000	0.0795	0.2780	0.0444
$n = 100, k = 33$	2000	0.0787	0.1970	0.0332
$n = 200, k = 68$	1000	0.0712	0.1430	0.0286



(a) A comparison of the relative frequency graphs for the best performing cases (excluding the interval $\{0\}$)



(b) Comparing the frequencies of zero performance

Fig. 5. Relative frequency results for top performing cases

Next we observe the apparent decrease in standard deviation from the mean for fixed k/n as n increases (see Fig. 5(a)). For each of the best performing cases, the decline in relative frequency of the "no-improvement" occurrence should be noted in Fig. 5(b); specifically, we observe that when $n = 10, k = 2$ there is an approximately 30% chance that our algorithm will perform no better than MSTH, whereas when $n = 50, k = 17$ that chance drops to about 1%.

VI. Conclusion and Future Work

The k-bottleneck Steiner tree problem has many applications in network theory and location science. The problem is especially significant for WSN deployment scenarios, especially those where all sensors transmit at more or less the same rate, and where *trees* will be constructed by the routing protocol. In this paper we presented a new heuristic for this NP-hard problem, and showed, via simulation, that our algorithm performs about 8% better (for some values of k) than the currently best performing heuristic.

One aspect that could be included in a generalisation of our WSN model is the potential for split routing. This would allow for the possibility that smarter routing protocols will be utilised by the network - for instance, the network may use a dynamic protocol that diverts traffic away from sensors with depleted batteries. Our assumption that all sensors transmit data at the same rate does not hold for all sensor networks, and we therefore highlight the importance of undertaking research on a generalised version of the bottleneck problem, i.e., one that incorporates flow.

References

[1] S. W. Bae, C. Lee, and S. Choi, "On exact solutions to the Euclidean bottleneck Steiner tree problem", *WALCOM* 2009, LNCS 5431, pp. 105–116, S. Das, R. Uehara (Eds.).

[2] M. Brazil, C. J. Ras, K. Swanepoel, and D. A. Thomas, Generalised k-Steiner tree problems in normed planes, submitted for publication.

[3] M. Brazil, C. J. Ras, and D. A. Thomas, Relay Augmentation for Lifetime Extension of Wireless Sensor Networks, submitted for publication.

[4] Z. Drezner and G. O. Wesolowsky, A new method for the multifacility minimax location problem, *J. Operational Research Society*, vol. 29, 1978, pp. 1095–1101.

[5] X. Du, X. Liu, and Y. Xiao, "Density-varying high-end sensor placement in heterogeneous wireless sensor networks", *IEEE Int. Conf. Commun.* 2009, pp. 1–6.

[6] D.–Z. Du, L. Wang, and B. Xu, "The Euclidean bottleneck Steiner tree and Steiner tree with minimum number of Steiner points", *COCOON* 2001, LNCS 2108, pp. 509–518, J. Wang (Ed.).

[7] G. Georgakopoulos and C. H. Papadimitriou, The 1-Steiner Tree Problem, *Journal of Algorithms*, vol. 8, 1987, pp. 122–130.

[8] S. Kwon, J. Kim, and C. Kim, "An efficient tree structure for delay sensitive data gathering in wireless sensor networks", *22nd Int. Conf. Advanced Information Networking and Applications*, March 2008, pp. 738–743.

[9] J.–S. Li, H.–C. Kao, and J.–D. Ke, Voronoi-based relay placement scheme for wireless sensor networks, *IET Commun.*, vol. 3, 2009, pp. 530-538.

[10] R. F. Love, G. O. Wesolowsky, and S. A. Kraemer, A multifacility minimax location method for Euclidean distances, *Int. J. Prod. Research*, vol. 11, 2009, pp. 37–45.

[11] M. Sarrafzadeh and C. K. Wong, Bottleneck Steiner trees in the plane, *IEEE Trans. Comput.*, vol. 41, 1992, pp. 370–374.

[12] L. Wang and D.–Z. Du, Approximations for a bottleneck Steiner tree problem, *Algorithmica*, vol. 32, 2002, pp. 554–561.

[13] F. Wang, D. Wang, and J. Liu, "Traffic-aware relay node deployment for data collection in wireless sensor networks, *Proc. 6th Annu. IEEE Commun. Soc. Conf. Sensor, Mesh and Ad Hoc Communications and Networks*, 2009, pp. 351–359.

[14] D. M. Warme, P. Winter, M. Zachariasen. Exact Algorithms for Plane Steiner Tree Problems: A Computational Study. In D.–Z. Du, J. M. Smith, J. H. Rubinstein, editors, *Advances in Steiner Trees*, pp. 81-116, Kluwer Academic Publishers, Boston, 2000.

[15] Y. Wu, S. Fahmy, and N.B. Shroff, "On the Construction of a Maximum-Lifetime Data Gathering Tree in Sensor Networks: NP-Completeness and Approximation Algorithm", *IEEE INFOCOM 2008*, Phoenix, AZ, April 2008.

[16] K. Xu, H. Hassanein, G. Takahara, and Q. Wang, Relay node deployment strategies in heterogeneous wireless sensor networks, *IEEE Trans. Mobile Comput.*, vol. 9, 2010, pp. 145–159.

[17] W. Yang, C. Zhang, and M. Li, "LBN: load-balancing network for data gathering wireless sensor networks", Embedded and Ubiquitous Computing, E. Sha et al. (Eds.): EUC 2006, LNCS 4096, pp. 204 - 213.