Wireless Sensor Network Aided Search and Rescue in Trails *

Peng Zhuang, Qingguo Wang, Yi Shang, and Hongchi Shi
Department of Computer Science
University of Missouri-Columbia
{pz797, qwp4b}@mizzou.edu, {shangy, shih}@missouri.edu

Bei Hua
Department of Computer Science and Technology
University of Science and Technology of China
bhua@ustc.edu.cn

ABSTRACT

In recent years, wireless sensor networks have been used in applications of data gathering and target localization across large geographical areas. In this paper, we study the issues involved in applying wireless sensor networks to search and rescue of lost hikers in trails and focus on the optimal placement of sensors and access points such that the cost of search and rescue is minimized. Particularly, we address two problems: a) how to identify the lost hiker position as accurately as possible, i.e., obtain a small trail segments containing the lost hiker; and (b) how to search efficiently in trail segments for different trail topologies and search agent capabilities. For the optimal access point deployment problem, we propose theoretical models that consider both efficiency and accuracy criteria and present analytical results for simpler trail topologies. For complicated graph topologies, we develop efficient heuristic algorithms with various heuristics. After access point deployment is decided, the actual cost of search in individual trail segment can be computed. We analyze four different types of search and rescue agents, present algorithms to find the optimal search pathes for each one of them, and compute their search costs. The algorithms are developed based on solving Chinese Postman problems. Finally, we present extensive experimental results to examine the accuracy of the mathematical models and compare the performances of different methods. A heuristic method, divide-merge, is shown to outperform all others and finds near-optimal solutions.

Categories and Subject Descriptors

I.2.11 [**Distributed Artificial Intelligence**]: Multiagent systems; C.2.1 [**Network Architecture and Design**]: Distributed networks; G.2.2.2 [**Graph Theory**]: Graph algorithms, Path and circuit problems

*Conference name:Conference infoscale 2007
*Version: 7.5.441
*Virus Database: 268.17.32/677 - Release Dage:2/8/2007 9:04 PM

General Terms

Algorithm; Design

Keywords

Search and rescue; Mobile agent; Wireless sensor network; Graph partitioning; Chinese postman problem

1. INTRODUCTION

In recent years, wireless sensor networks have been applied to many data gathering and target localization and tracking applications over large geographical areas [8, 9].These sparsely deployed sensor networks maintain connectivity through the use of mobile agents that help transfer data from sensors to access points so as to increase communication bandwidth and reduce power consumption at sensor nodes.

One representative example of sparse sensor networks is CenWits (**C**onnection-less S**en**sor-Based Tracking System Using **Wit**ness**s**es), which is proposed for search and rescue of subjects (e.g. people or wild animals) in emergency situations in wilderness areas [8]. In CenWits, each hiker wears a battery-powered small device (called sensor nodes) with integrated computation, wireless communication, and positioning (via GPS) capabilities. Access points (APs) are placed at popular locations like the trail heads/ends, intersections, scenic view points, or resting/camping areas. When two hikers with their sensor nodes come close enough to establish a wireless communication, they store each other's presence as a witness record and also exchange previous witness records. When a hiker with a sensor node enters the range of an AP, the presence of the hiker is recorded. In addition, the sensor node uploads all its witness records to the AP. APs are connected to a processing center via another long-range wireless network (e.g. a satellite network) so that the witness information is transmitted to the processing center to be used for various applications, such as search and rescue lost hikers.

While it provides a general framework and prototype system, CenWits does not consider the optimization problem of minimizing the search cost given a limited number of APs, or given the maximal tolerated search cost, what is the minimal number of APs need in a trail. This is important because the AP equipped with long range (several miles) communication is much more expensive than a sensor node, which make it unfeasible to deploy a dense AP network.

In this paper, we study several issues raised in the design of CenWits-

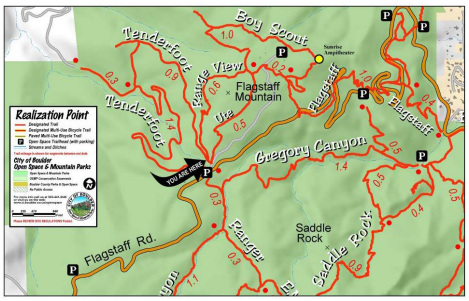

Figure 1: A map for trails outside Boulder, Colorado.

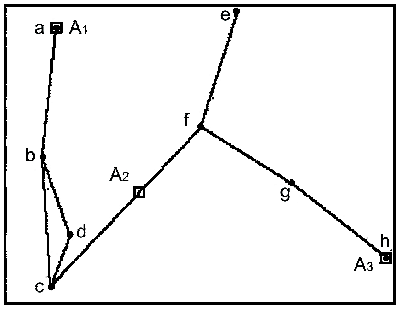

Figure 2: An example of AP placement in a trail graph. The three squares represent APs.

like systems for finding lost hikers. Particularly, we address two problems: a) how to identify the lost hiker position as accurately as possible, i.e., obtain a small trail segments containing the lost hiker; and (b) how to search efficiently in trail segments with different topologies. The system is a multi-agent system in which hikers are mobile agents. Together with APs, they form a dynamic and partially connected network that supports cooperative agent tracking. Furthermore, we propose methods for optimal search path planning for a single or multiple search and rescue agents.

The paper is organized as follows. In Section 2, we present problem formulation. We define two objectives, search cost and location uncertainty, and discuss their trade-offs. In Section 3, we solve the AP placement problem based on the location uncertainty objective. We derive theoretical results on simpler trail topologies and present efficient heuristic algorithms for more complicated trail topologies. In Section 4, we discuss four different types of search and rescue agents, present algorithms to find the optimal search pathes for each one of them given a trail segment, and compute their search costs. In Section 5, we present experimental results to examine the accuracy of the mathematic models and compare the performances of different methods. Finally, we discuss related work in Section 6 and summarize the paper in Section 7.

2. PROBLEM STATEMENT

A typical scenario of finding a lost hiker using the wireless sensor network system is as follows. A hiker's movement is recorded by other hikers and APs as witness records. The lost case is assumed to be non-moving accident, such as being injured, sick, or stuck, along the trial. When a hiker is determined to be lost, e.g., based on the time the hiker has not been witnessed, the hiker's lost region is decided based on the witness records for him, which is a trail segment. One or more search and rescue agents are sent out to the trail segment to find the lost hiker.

We represent the trails as a weighted undirected planar graph $G = (V, E)$ with vertices for intersections, edges for direct paths, and edge weights $w(e), e \in E$ for distances. Let l denotes the total weights of the graph where $l = \sum_{e \in E} w(e)$. Fig. 1 shows an example of a trail map near Boulder, Colorado.

Suppose m APs ($\{A_1, A_2 ... A_m\}$) are deployed along the trails. They divide the graph into non-overlapping subgraphs called *trail segments*. A trail segment is bounded by APs and denoted by $G_i = (V_i, E_i)$, $i \in [1, m_s]$, where m_s is the total number of trail segments. Assuming the sensor network system can determine the hiker's moving direction. Once a hiker passes an AP, we know which trail segment he is entering and thus the location estimate is narrowed down to the trail segment rather than the entire trail map. An example is shown in Fig. 2 with two trail segments $\{(A_1, b)$ (b, c) (b, d) (c, d) $(c, A_2)\}$ and $\{(A_2, f)$ (f, e) (f, g) $(g, A_3)\}$.

The probability of a hiker getting lost in a trail segment can be derived as follows. Let $w_i, i \in [1, m_s]$ denotes the total weight of G_i. Suppose the hiker has an equal chance to get lost at any point along the trail, i.e., the lost probability is uniform in the trails. Let $p(G_i)$ be the probability of a hiker being lost in segment G_i, we have

$$p(G_i) = \frac{w_i}{l} \tag{1}$$

Let $c(G_i)$ denotes the search and rescue operation cost function in G_i. The expected search cost of all trail segments is

$$C = \sum_{i=1}^{m_s} p(G_i)c(G_i) = \sum_{i=1}^{m_s} \frac{w_i \bullet c(G_i)}{l} \tag{2}$$

Thus, the problem of deploying APs such that the expected cost of finding a lost hiker is formulated as follows:

$$\min C \text{ subject to } \sum_{i=1}^{m_s} w_i = l \tag{3}$$

Under different scenarios, we may have different definitions of $c(G_i)$. For example, some applications ask for minimizing the average search cost, whereas others may impose a maximal search cost it can afford. We use expected search cost $c^E(G_i)$ for the first case and maximal search cost $c^M(G_i)$ for the second case. If G_i is as simple as a single edge, the expected search cost and maximal search cost is simply $c^E(G_i) = \frac{w_i}{2}$ and $c^E(G_i) = w_i$. However, for a trail segment of a general graph, depending on how the different search agents operate, such as ground vs. air, the search costs will be different and computing their exact costs can be expensive. We address the issues in more details later in Section 4.

To simplify the problem, we use another metric, *location uncertainty*, in place of the search cost C, which can be computed in linear time. Let Y_i denote the location at which the hiker is lost in trail segment G_i. Let σ_i be the standard deviation of Y_i. Assume Y_i follows a uniform distribution, we have

$$\sigma_i = \frac{1}{2\sqrt{3}} w_i \tag{4}$$

Location uncertainty, U, is defined based on the expected value of

the standard deviation

$$U = \sqrt{3}E(\sigma_i) \tag{5}$$

Now, the new objective is to find an AP placement that minimizes U. Specifically, we solve the following problem

$$\min U \text{ subject to } \sum_{i=1}^{m_s} w_i = l \tag{6}$$

In our approach of solving the AP deployment problem, i.e., placing APs so that the expected cost of finding a lost hiker by a particular type of search and rescue agent is minimized, we divide the problem into two simpler sub-problems: solving Eq. (6) and then minimizing $c^E(G_i)$ and $c^M(G_i)$ for a given trail segment G_i. This approach is much more efficient than solving the original problem in Eq. (3) directly, which requires expensive calls to compute either $c^E(G_i)$ or $c^M(G_i)$. In contrast, U can be computed efficiently and also enables theoretical analysis. Empirically, the results of using U are very close to those of using C, as shown in our experimental results in Section 5.

3. ACCESS POINT PLACEMENT

Based on the trail structure, we divide the problem into two categories: a simple edge or a graph. The single hiker case is investigated first, followed by the multiple hikers case.

3.1 Trails of a single path

We start with a simple case: a single path with two trail heads A_1 and A_m. Let the trail be l miles. m APs (A_1, A_2, ..., A_m) are depolyed along the trail. To determine whether it contains a hiker, each trail segment needs two APs on its two end points. Therefore, with two AP deployed on the two trail heads, $m_s = m - 1$. From Eqs. (4) and (1), we have

$$U = \sqrt{3}E(\sigma_i) = \sqrt{3}\sum_{i=1}^{m_s} \sigma_i p(G_i) = \frac{1}{2l}\sum_{i=1}^{m_s} w_i^2 \tag{7}$$

From Eq. (7) and by means of Lagrange function, we can solve the optimization problem in Eq. (6). The location uncertainty U has the minimal value if and only if $w_i = \frac{l}{m_s}$ for all $i \in [1, m_s]$. Therefore, in a single path scenario, the optimal solution is to place the access points evenly along the trail.

3.2 Trails of a graph

For a trail segment G_i with total length w_i, we have

$$\sigma_i = \frac{w_i}{2\sqrt{3}} = \frac{\sum_{e\in E_i} w(e)}{2\sqrt{3}},\ p(G_i) = \frac{w_i}{l} = \frac{\sum_{e\in E_i} w(e)}{l} \tag{8}$$

The uncertainty U is

$$U = \frac{1}{2l}\sum_{i=1}^{m_s} w_i^2 = \frac{1}{2l}\sum_{i=1}^{m_s} (\sum_{e\in E_i} w(e))^2 \tag{9}$$

Let $F = \{w_i | i \in [1...m_s]\}$, then

$$Var(F) = E(w_i^2) - (E(w_i))^2 = \frac{\sum w_i^2}{m_s} - (\frac{\sum w_i}{m_s})^2 \tag{10}$$

Thus, we can simplify U as

$$U = \frac{m_s}{2l}E(w_i^2) = \frac{m_s Var(F)}{2l} + \frac{l}{2m_s} \tag{11}$$

When the number of APs, m, is given, if m_s is fixed (like a single path where $m_s = m-1$), U is minimal when $Var(F) = 0$, which means the APs are placed at the same interval. However, when m_s is not fixed, seeking a even placement may not result in an optimal solution. Generally speaking, a closed form solution does not exist for Eq. (11), and we can only use computational methods to find the optimal solution. The optimal solution balances between m_s and $Var(F)$.

For graph topologies, we propose a two-phase method to solve the AP placement problem. In the first phase, the optimal solution of deploying APs on a subset of the vertices is sought; In the second phase, the solution is improved by allowing the APs to be moved to edges. The method is presented in the next two sub-sections.

3.3 AP placement on vertices

The objective is to find a subset V' of m vertices within V that partitions G into m_s disjoint subgraphs such that the value of Eq. (9) is minimal. Enumerate all combinations requires $O(|V| \bullet C_m^{|V|})$ time. Here, we present four efficient methods:

1) Local Search. It minimizes U via a typical local search technique. It starts with a random AP placement such that $A_1, A_2...A_m$ are at distinct vertices. At every iteration, it tries to move one AP to a neighboring vertex with the maximal reduction on the uncertainty U. It stops as soon as it reaches a local optimal. The time complexity is $O(|V|^2)$. In our experiments, we observed that the search space of U is very rugged with many local minimal. Thus, the performance of local search is not very good.

2) Min-max Local Search. It tries to minimize $max(w_i)$. The search space of $max(w_i)$ is smoother and the local search algorithm usually runs more iterations than method 1 (7.5 vs. 1.8 on average on our test cases). In addition, it also helps to reduce $Var(F)$, which is proportional to U according to Eq. (11). It usually finds better solutions than method 1. The time complexity is $O(|V|^2)$.

3) Divide and Merge Algorithm. It starts with a full AP placement where each vertex has an AP. At every iteration, it removes an AP that results in the minimal increase of U. After $|V| - m$ iterations, it stops with a solution of m APs. The time complexity is $O(|V|^2)$.

4) Hyper-edge K-partitioning (HKP) Algorithm. The graph partitioning problem, in general, is studying how to achieve k balanced partitions and the total number of edges connecting any two adjacent partitions is minimal. We exchange the vertices and edges in the trail map so that we can adopt existing algorithms. The transformed graph is no longer a simple graph and therefore we need to use the algorithms designed for hypergraph.

We first transform the trail map G into a hyper graph $G^h = (V^h, E^h)$ where the vertices and edges are exchanged, and the weight of the vertices in G^h are the edge weight in G. Let $|E_c^h|$ denotes the total number of edges connecting any two adjacent partitions. The problem becomes

$$\min |E_c^h| \text{ subject to } \forall i \in [1...m_s]\ \ w_i \leq \alpha \bullet \frac{l}{k} \tag{12}$$

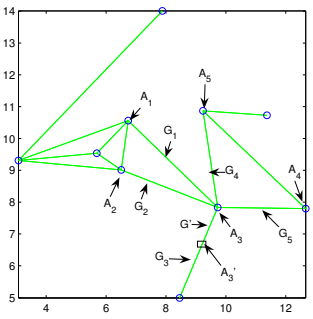

Figure 3: An example of moving an AP, A_3, from a vertex to an edge point, A_3'.

where l is the total weight of G^h, w_i is the weight of partition G_i^h, and α is a tolerance factor usually set to 1.05.

Next, we use the algorithm proposed in [10] based on greedy graph growing partitioning to obtain the partition on the transformed graph. The algorithm starts with $|V^h|$ partitions each of which containing only one vertex of the hypergraph. At every iteration, the algorithm selects a random partitions whose size is smaller than $\alpha \bullet \frac{l}{k}$ and joins the partition with an adjacent partition that results in the minimal increase or maximal decrease in $|E_c^h|$. Usually t multiple trails are need to obtain a reasonable good solution but $t \ll |V^h|$ ([11]). The algorithm requires $O(t \bullet |V^h|^2)$ tries where $|V^h|$ is the number of vertices in the hypergraph. Because the original trail graph is planar,we have $|V^h| = O(|V|)$, and because each try calls the subroutine of computing U whose time complexity is $O(|V|)$, the overall complexity is $O(|V|^3)$.

Finally, we exchange the vertices and edges again to get the AP placement on the original trail map.

3.4 Edge refinement - moving APs to edges

The solution of any of the methods presented in the last sub-section restricts the placement of APs to vertices. In this section, We improve the solution by moving APs onto edges.

Given an initial placement of APs, consider an AP, $A_i \in V$, and the set Ω of all adjacent trail segments $\{G_j = (V_j, E_j) | \, V_j \cap \{A_i\} \neq \phi\}$. We have

$$W = \sum_{G_j \in \Omega} \sum_{e \in E_j} w(e) \tag{13}$$

Assume we can improve the solution by moving A_i to one of the adjacent trail segment G_j. Let G' denotes the trail segment enclosed by the new A_i position and the original position. An example is shown in Fig. 3. Let $w_j = \sum_{e \in G_j} w(e)$, $w' = \sum_{e \in E'} w(e)$ and $k = |\Omega|$. When we move A_i to one edge, there are two and only two trail segments adjacent to A_i's new position. Let G^0 and G^1 denotes the two new trail segments with weights w^0 and w^1, respectively. The uncertainty of the new placement is

$$U' = \frac{1}{2l}(w^0 + w^1) = \frac{1}{2l}[(\sum_{x \in \{1 \ldots k\} - \{j\}} w_x + w')^2 + (w_j - w')^2], \tag{14}$$

whereas the uncertainty of the original placement is

$$U = \frac{1}{2l} \sum_{x \in \{1 \ldots k\}} w_x^2 \tag{15}$$

The difference between the new placement and the original placement is

$$\Delta U = U' - U = \frac{1}{2l}(w'^2 + (W - 2w_j)w' + \sum_{x \neq y}^{x,y \in \{1 \ldots k\} - \{j\}} w_x w_y) \tag{16}$$

Take the derivative of Eq. (16) and set it to 0, we can compute the w' value for the minimal uncertainty (if exists), which is

$$w' = w_j - \frac{W}{2} \quad \text{for} \quad w_j \geq \frac{W}{2} \tag{17}$$

There are two important properties of the new placement found by Eq. (17): the new placement is both half-divided and unique. They are important in proving the correctness of the algorithm. (The proofs are omitted due to the space limit.)

THEOREM 3.1. *For a subgraph G^- of G composed of an access point v and its adjacent trail segments, the new location of access point v' found by Eq. (17) divides G^- into two equal graphs G_1^-, G_2^- with $w(G_1^-) = w(G_2^-)$ where $w(G_x)$ is the sum of all edge weights of graph G_x. We say that v'* half-divides *the graph G^-.*

THEOREM 3.2. *There is at most one new placement found by Eq. (17). The solution is* unique.

According to Theorem 3.1, the new location can be determined by examining whether there is a place that half-divides the trail segment. In addition, Theorem 3.2 guarantees the new placement we found is unique and local-optimal.

The algorithm is developed based on the theorems is as follows.

1. Start with an initial placement.
2. At every iteration, move one AP in order to achieve the maximal reduction in the overall uncertainty.
3. Stop when no further improvement is possible.

In practice, the improvement quickly becomes insignificant after a few iterations. Thus we set the maximal number of iterations to be relatively small, e.g., 10.

3.5 Multiple Hikers

When there are multiple hikers coming across each other, they provide more location information of each other. Let x be the number of other hikers that a hiker h runs into before he gets lost. For a single path trail, if the hikers are randomly distributed in the trail, then the average uncertainty of the lost hiker location is

$$\overline{U} = \frac{l}{m_s + x + 1} \tag{18}$$

For graph topologies, it is

$$\overline{U} \approx \frac{l}{m_s + \frac{m_s}{m} \bullet x} \tag{19}$$

The detailed derivations are omitted due to space limit.

Given the expected number of hikers on each trail segment, we use Eq. (19) with local search techniques to improve the AP placement solution found in the single hiker case. The result is that fewer APs are deployed in trail segments containing more hikers and more APs in the ones with fewer hikers to achieve smaller $\overline{U}$.

4. SEARCH AND RESCUE

Once an AP placement solution is found and the trail segment containing the lost hiker is identified, search and rescue agents are sent to find the lost hiker with the lowest cost, e.g., shortest amount of time. The problem of minimizing the maximal or expected search cost is equivalent to finding an optimal search path in a graph. Let's take a look of the following four types of search and rescue (SaR) agents that result in different definitions of the search cost.

1. A single ground SaR agent (S-GSA): A single rescue team using a ground vehicle capable of traveling along the trails only.
2. Multiple ground SaR agents (M-GSA): A number of ground rescue teams perform the SaR mission separately.
3. A single air SaR agent (S-ASA): A single rescue team using an aerial vehicle capable of traveling along the trails and jumping from one trail branch to another.
4. Multiple air SaR agents (M-ASA): Several air SaR agents operate separately.

The SaR mission consists of two steps. First, the agents take a shortest path to the trail segment. Then they take the shortest search trajectory based on their capabilities to scan through all the edges belonging to trail segment.

4.1 Single ground SaR agent (S-GSA)

In the worst case, a single ground SaR agent travels all edges of the trail segment G_i at least once and the cost is:

$$c^M(G_i) = c'(P) + \sum_{e \in E_i} n(e)c(e) \tag{20}$$

where E_i is the set of all edges in G_i, $n(e)$ is the times an edge is visited, $c(e)$ is the cost to search the edge, and $c'(P)$ is the cost to travel to G_i along shortest path P. Assume P is fixed, then this problem is a Chinese Postman Problem (CPP) [12] on an undirected graph. There exist polynomial time algorithms to find the optimal solution. Let w_i denotes the total weight of G_i. The maximal travel length is between w_i (when G_i is an Euler tour) and $2w_i$ (when G_i is a tree).

While the CPP algorithm helps to minimize the maximal search cost by constructing an Euler path, minimizing the expected search cost is more important in practical sense. In order to compute the expected search cost $c^E(G_i)$, we categorize the traveled path in G_i into two types: tour segments and redundant segments. A tour segment is a path $(e_1, e_2, ..., e_n)$ whose edges have not been visited before. A redundant segment is a path whose edges have all been visited for at least once. For simplicity, we use the edge weight as the search cost in the rest of the discussion. The total weight of all tour segments is w_i and the total weight of all redundant segments represents the overhead cost. Let $l_t, (t = 1, ..., n)$ denotes the n tour segments. Let $r_j, (j = 1, ..., (n-1))$ denotes the $n-1$ redundant segments. The search path is a sequence of segments alternating between tour segments and redundant segments $(l_1, r_1, l_2, r_2, ..., r_{n-1}, l_n)$. Let $w(l_t)$ denotes the weight of tour segment l_t and $w(r_j)$ denotes the weight of redundant segment r_j, the expected search cost when the lost hiker is found in tour segment l_t is

$$c^E(l_t) = c'(P) + W^{b(l_t)} + \frac{w(l_t)}{2} \tag{21}$$

where $W^{b(l_t)} = \sum_{j=2}^{t} (w(l_{j-1}) + w(r_{j-1}))$ is the total weights of the path traveled before reaching the tour segment l_t, and $\frac{w(l_t)}{2}$ is the expected search distance in segment l_t. Let $p(l_t)$ denotes the probability of a hiker being lost in segment l_t. Assume $p(l_t)$ follows uniform distribution in G_i,

$$p(l_t) = \frac{w(l_t)}{w_i} \tag{22}$$

Therefore, the expected search cost in G_i is

$$c^E(G_i) = \sum_{t=1}^{n} p(l_t)c^E(l_t) \tag{23}$$

$$= c'(P) + 0.5w_i + \sum_{t=1}^{n} \frac{w(l_t)}{w_i}[\sum_{j=2}^{t} w(r_{j-1})] \tag{24}$$

The detailed derivation is omitted due to space limit.

When G_i is an Euler path and we start from an end point, we only have one tour segment and $c^E(G_i) = c'P + 0.5w_i$.

We propose a heuristic method to reduce the expected search cost that performs a greedy walk on the Euler graph found by the CPP algorithm. The method always chooses unvisited edges when possible. This is based on the observation from Eq. (24) that the early tour segments have less previous redundancy $\sum_{j=2}^{t} w(r_{j-1})$ and the increase of their lengthes results in higher $p(l_k) = \frac{w(l_t)}{w_i}$. Further improvements can be made by a local search process that exchanges edges between adjacent tour segments and redundant segments. Our experimental data show that the heuristic method outperforms the classic Fleury's algorithm with respect to achieving lower expected search cost.

4.2 Multiple ground SaR agents (M-GSA)

With multiple search agents, the problem becomes Min-Max k-Chinese Postman Problem (MM k-CPP), in which $k, k \geq 2$, postmen have to search the graph with a search cost for each edge and the maximal search cost for any agents is minimal, which is given by

$$c^M(G_i) = c'(P) + \max_{x=[1...k]} \left(\sum_{e \in E_i^{T(x)}} n(e)w(e) \right) \tag{25}$$

where $E_i^{T(x)}$ is the set of edges travelled by agent x in G_i. A solution can be found using a fairly recent algorithm [1] that utilizes the principle of tabu search.

The expected search cost for M-GSA is

$$c^E(G_i) = c'(P) + \frac{w_i}{2} + \max_{x=[1,\ldots,k]} \left(\sum_{t=1}^{n^x} \frac{w(l_t^x)}{w_i} [\sum_{j=2}^{t} w(r_{j-1}^x)] \right) \tag{26}$$

where $l_1^x, ..., l_{n^x}^x$ and $r_1, r_2, ..., r_{n^x-1}$ are the n^x tour segments and the $(n^x - 1)$ redundant segments traveled by agent x, respectively. We use the same greedy heuristic method as in the single ground SaR agent scenario to plan the search trajectory.

If we have very large number of ground SaR agents to search every branches in parallel, then breath-first search (BFS) optimizes both maximal and expected search cost. Assume BFS starts from vertex u. We have the search cost

$$c^M(G_i) = c'(P) + w(L_{i,u}) \tag{27}$$

where $L_{i,u}$ is the longest path in G_i starting from u. If G_i is a tree, $w(L_{i,u})$ simply represents the height of the tree. If G_i contains at least one cycle, the search may end at the middle of some edge. For example, if G_i is a simple cycle, there are two branches at u and the two search agents separate and meet again when they all travel $\frac{w_i}{2}$ distance. The final meeting point may be in the middle of some edge and in this special case, $w(L_{i,u}) = \frac{w_i}{2}$.

The expected travel time of BFS is

$$c^E(G_i) = c'(P) + 0.5w(L_{i,u}) \tag{28}$$

When u is fixed, we use BFS to find $w(L_{i,u})$ and the algorithm runs in linear time ($O(|E_i|)$) where E_i is the set of edges in G_i.

4.3 Single air SaR agent (S-ASA)

The problem of minimizing the search cost of a S-ASA is quite similar to that of a single ground agent except that a S-ASA is capable of crossing from one trail to another. Assume the crossing is only allowed between any two vertices, then we can transform the problem to classic Rural Postman Problem (RPP) by adding dummy edges in the original G_i to obtain the complete graph K_i. We categorize the edges in K_i into required edge set E^{K_i+}, whose edges have a corresponding edge in G_i, and non-required edge set E^{K_i-}, whose edges are all dummy edges. The objective of RPP is to visit all edges in E^{K_i+} with the minimal cost, which is

$$c^M(G_i) = c'(P) + \sum_{e \in E_i^T} n(e)w(e) \tag{29}$$

where E_i^T is the set of all edges (include dummy edges) traveled. Eq. (29) can be computed using existing algorithms, such as the one based on local search in [5].

In order to compute the expected search cost, we re-define a redundant segment as a path whose edges have either been visited before or are dummy edges in E^{K_i-}. The expected cost is given by

$$c^E(G_i) = c'(P) + \frac{w_i}{2} + \sum_{t=1}^{n} \frac{w(l_t)}{w_i} [\sum_{j=2}^{t} w(r_{j-1})] \tag{30}$$

which is similar to Eq. (24). The method proposed for S-GSA can also be applied here.

4.4 Multiple air SaR agents (M-ASA)

Performing the same graph transformation on G_i as in single air SaR agent, we can solve the problem as the Min-Max k-RPP. The maximal cost and the expected cost follow the same definition as in Eq. (25) and Eq. (26). Solving the problem of minimizing maximal search cost is similar to solving the MM k-CPP in the sense that they both use local search techniques to obtain solutions based on heuristics for a single agent case. We use the same greedy method as for the other scenarios to find a search path with minimal expected cost.

5. EXPERIMENTAL RESULTS

In the first set of experiments, we show that the objective of location uncertainty U is a good approximation to the real search cost C as in Eq. (2). Random trail maps are generated by placing 10 vertices randomly inside a square region of size 50×50. Nearby vertices are first connected by edges with the lengthes uniformly distributed in region $(0, 5)$. When two edges cross over, one is selected to be removed probabilistically. The result is a connected planar graph with the maximal node degree no more than 5. For the single hiker and single ground SaR agent case, we ran different algorithms with varying number of access points. Each data point is the average of 30 runs of random instances.

Using the results of the divide-merge (DM) algorithm, Fig. 4(a) compares their location uncertainty value U, half of the maximal search cost $C^M/2$, and the mean search cost C^E. U is a lower bound of $C^M/2$ and C^E since it does not include the redundant search segments. $U = C^E = \frac{C^M}{2}$ if and only if the non-redundant trail segments form an Euler path. All three curves show the same decreasing trend as the number of access points increases. This is because more APs divide the trails into more trail segments.

Using the results of divide-merge (DM) and min-max local search (MMLS), we again show that U is effective and its values are very close to those of $C^M/2$ and C^E as in Fig. 4(b). The y-axis is the normalized difference between the MMLS results and the corresponding DM results:

$$y = \frac{Q_{MMLS} - Q_{DM}}{Q_{DM}} \tag{31}$$

where Q_{MMLS} denotes the average value of MMLS results from 30 runs and Q_{DM} denotes that of DM. As we can observe from the figure, the three performance metrics show similar increasing trend toward 5, which is half of the total vertex number $|V|$, and start to decrease afterwards. When the number of access points m is larger than 70% of $|V|$, the two algorithms show little difference for all the three metrics. In the rest of the experiments, we use U as the performance metric for its simplicity and efficiency.

In the next set of experiments, we compare the results of five algorithms: divide-merge (DM), local search (LS), min-max local search (MMLS), the optimal solution of placing APs on vertices only, and the divide-merge with refinement (DMR). The first four algorithms restrict the placement of APs to vertices, whereas the last one can place APs on edges.

Fig. 4(c) shows the location uncertainty values of the results found by the five algorithms for different number of access points. Local

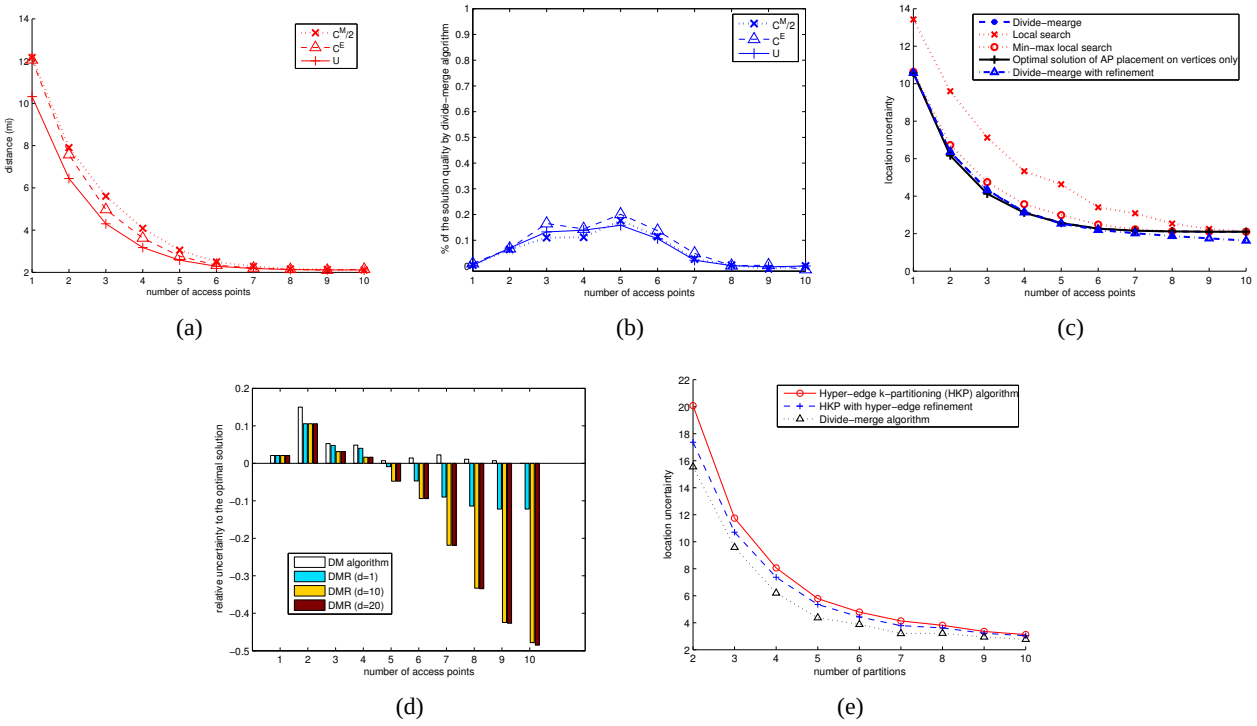

Figure 4: (a) Comparison of values of three difference metrics, U, $MAX(C)/2$, and $E(C)$, on the solutions of DM. (b) The differences of U, $MAX(C)/2$, and $E(C)$ values between MMLS and DM normalized over DM solutions, respectively. (c) Comparison of four methods for placing AP on vertex and DM with edge refinement that allows APs being placed on edges. (d) Comparison of DM without and with edge refinement of different iterations. (e) Comparison of DM, HKP, and HKP with hyper edge refinement.

search is the worst, while the others are similar. Between the two local search algorithms, MMLS achieves better solution by reducing the variance of the edge lengths of all trail segments ($Var(F)$ in Eq.(11)) and producing a smoother search landscape that allows more hill-climbing iterations (7.5 iterations of MMLS vs. 1.8 iterations of LS on average).

The solutions of DM and MMLS are very close to the optimal solutions and DM is slightly better. Furthermore, DM is scalable and runs the fastest, very fast even when the number of vertices is large. In addition, the data shows a trend that the more APs deployed, the lower the location uncertainty is. The curves become flat after $m \geq 70\% \times n$. It corresponds to the percentage of the deploying an AP on a degree 1 vertex will not reduce the uncertainty because it is already the boundary of a connected component. To the contrary, if we relax the constraint of vertex-only, the AP on the 1-degree vertices can be moved to achieve lower uncertainty. This is why the curve of the solutions with edge refinement by DMR continue to improve after $m = 7$.

The benefit of edge-refinement in DMR is significant when the number of APs is large, as shown in Figs. 4(c) and 4(d). In Fig. 4(d), the solutions of DM and DMR with different iterations (1, 10, or 20) are normalized over the corresponding optimal vertex-only solutions. The data shows that DMR improves DM even with only one iteration. Running the edge-refinement for 10 iterations is significantly better than running for one iterations, but is almost the same as running for 20 iterations. When the number of APs is over 5, DMR achieves solutions better than the optimal solutions of vertex-only placement.

Finally, we compare divide-merge (DM) with the hyper-edge k-partitioning algorithm (HKP) and HKP with Hyperedge Refinement (HER). We implement the hyperedge refinement technique presented in [10], which is one of the two widely used refinement methods in hyper-graph partitioning problem (The other one is the FM algorithm [3]). We choose HER for its simplicity and good performance.

Random trail graphs in larger square regions with size 150×150 and 30 vertices were used in the experiments. The lengths of the edges are still in region $(0, 5)$. For different number of partitions ($k = 1, ..., 10$), HKP or HKP+HER returns the smallest number of APs it needs to achieve the partition. Using the same number of APs, DM returns its solution. Then we compare the location uncertainty of their solutions. Fig. 4(e) shows that DM is consistently better than HKP+HER, which in turns is better than HKP.

6. RELATED WORK

(1) Access point placement. The graph partitioning problem is defined as to divide the graph into k roughly equal disconnected subgraphs, such that the number of edges (or the sum of the edge costs) between different parts are minimized. It has been a popular issue in the field of scientific computing, VLSI design, and task scheduling. It is known NP-complete but many approximation algorithms have been proposed to find a reasonable good solution. One class of the algorithms is called the spectral partitioning [6], which are quite expensive. Another class uses the geometric information of the graph (if the coordinates of the vertices are known) [13]. The geometric methods usually run faster than the spectral method but

the solution is usually worse. The third class of methods is called the multilevel method. It first reduces the size of the graph (usually by collapsing the vertices and edges to yield a smaller graph), then finds a solution in the reduced graph and finally maps the solution to the original graph. The phase of mapping the solution back is also referred as uncoarsen phase when a refinement is usually adopted [11, 7]. The multilevel approach usually provides better solution than spectral methods at lower cost [15].

(2) Search and rescue path planning. The problem can be formulated as the Chinese Postman Problem (CPP) on an undirected graph [12]. It tries to solve the problem on how to travel all graph edges with the minimal travel distance. A polynomial time optimal algorithm is available [2]. Another problem related to CPP is Rural Postman Problem (RPP) [14], where some of the edges are not required to be visited. The edges are divided to two sets, required or non-required. The objective becomes finding a shortest tour that travels the required edges. The problem is proven to be NP-hard [14]. Recently some heuristic methods based on either local search or Monte Carlo principles are proposed for RPP[5, 4].

7. CONCLUSION

In this paper, we study the problem of search and rescue (SaR) of lost hikers along trails with the help of wireless sensor networks. The goal is minimizing the expected or maximal search cost. We address the problem by dividing it into two simpler problems. First, we present theoretical analysis and propose efficient algorithms to the optimal AP placement problem. This helps to reduce the search and rescue operation to one trail segment. Then we analyze different SaR scenarios and propose methods to minimize the search cost. The maximal search cost is minimized using existing Chinese Postman Problem algorithms and the expected search cost is minimized using a new heuristic method. In simulation, we compare different algorithms and show that the solution quality obtained by an efficient heuristic method, divide-merge, are very close to the optimal solution.

In the future work, we plan to study the SaR operations where more than one type of SaR agents present. It is interesting because heterogenous SaR agents present both different search costs and search gains (e.g. the probability an agent will discover the lost subjects when they are in its sight). Furthermore, we plan to study the case of unknown hiking direction in the optimal AP placement problem. It presents challenges both in that a trail segment is not closed any more and in that the missing probability is no longer uniform along the trail.

8. ACKNOWLEDGEMENT

This work is partially supported by NSF Grant CNS-0423386 and a Chinese Fund for Foreign Scholars in University Research and Teaching Programs.

9. REFERENCES

[1] D. Ahr and G. Reinelt. A tabu search algorithm for the min-max k-chinese postman problem. *Comput. Oper. Res.*, 33(12):3403–3422, 2006.

[2] J. Edmonds and E. Johnson. Euler tours and the chinese postman problem. *Mathematical Programming*, 5:88–124, 1973.

[3] C. M. Fiduccia and R. M. Mattheyses. A linear-time heuristic for improving network partitions. In *DAC '82: Proceedings of the 19th conference on Design automation*, pages 175–181, Piscataway, NJ, USA, 1982. IEEE Press.

[4] G. G and L. G. A branch and cut algorithm for the undirected rural postman problem. *Mathematical Programming*, 87:467–481, 2000.

[5] G. Groves and J. van Vuuren. Efficient heuristics for the rural postman problem. *ORiON*, 21(1):33–51, 2005.

[6] B. Hendrickson and R. Leland. An improved spectral graph partitioning algorithm for mapping parallel computations. *SIAM Journal on Scientific Computing*, 16(2):452–469, 1995.

[7] B. Hendrickson and R. W. Leland. A multi-level algorithm for partitioning graphs. In *Supercomputing*, 1995.

[8] J. H. Huang, S. Amjad, and S. Mishra. Cenwits: A sensor-based loosely coupled search and rescue system using witnesses. In *Proceedings of ACM SenSys*, 2005.

[9] P. Juang, H. Oki, Y. Wang, M. Martonosi, L. Peh, and D. Rubenstein. Energy-efficient computing for wildlife tracking: design tradeoffs and early experiences with zebranet. In *ASPLOS*, 2002.

[10] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar. Multilevel hypergraph partitioning: Applications in VLSI domain. Technical report, 1997.

[11] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20(1):359–392, 1998.

[12] M.-K. Kwan. Graphic programming using odd or even points. *Chinese Math*, 1962.

[13] G. L. Miller, S.-H. Teng, and S. A. Vavasis. A unified geometric approach to graph separators. In *Proceedings of the 32nd annual symposium on Foundations of computer science*, pages 538–547, Los Alamitos, CA, USA, 1991. IEEE Computer Society Press.

[14] C. S. Orloff. A fundamental problem in vehicle routing. *Networks*, 4:35–64, 1974.

[15] A. Strehl. *Relationship-baased Clustering and Cluster Ensembles for High-dimensional Data Mining*. PhD thesis, University of Texa at Austin, 2002.