WIRELESS SENSOR NETWORK AIDED SEARCH AND RESCUE IN TRAILS

A Thesis

presented to

the Faculty of the Graduate School

University of Missouri-Columbia

In Partial Fulfillment

of the Requirements for the Degree

Master of Sciense

---

by

PENG ZHUANG

Dr. Yi Shang, Thesis Supervisor

December 2006

The undersigned, appointed by the Dean of the Graduate School, have examined the thesis entitled

WIRELESS SENSOR NETWORK AIDED
SEARCH AND RESCUE IN TRAILS

presented by Peng Zhuang

a candidate for the degree of master

and hereby certify that in their opinion it is worth acceptance.

_____

Dr. Yi Shang

_____

Dr. Hongchi Shi

_____

Dr. Zhihai He

ACKNOWLEDGMENTS

First and foremost, I would like to thank Dr. Yi Shang, my advisor for his continuous support to my research. During our weekly individual meeting, he provides great inspirations and guidance that help to make my research complete. I really learn a lot from him and am very impressed by his broad scientific insight.

I also want to thank all the people in our wireless sensor network research group, especially Qingguo Wang. Parts of the work in this research was started by him. He also provided valuable ideas for me to choose the research directions.

I would like to thank Dr. Hongchi Shi and Dr. Zhihai He for reviewing my thesis. I also want to thant Dr. Shi for his valuable suggestions to my research.

Finally, I would like to thank my family for always supporting me, especially my wife, who always believes in me.

WIRELESS SENSOR NETWORK AIDED SEARCH AND RESCUE IN TRAILS

Peng Zhuang

Dr. Yi Shang, Thesis Supervisor

ABSTRACT

In recent years, wireless sensor networks have been used in applications of data gathering and target localization across large geographical areas. In this thesis, 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 search region containing the lost hiker; and (b) how to search efficiently in search regions for different trail topologies and search agent capabilities.

We study the problem of achieving smaller search regions with different problem attributes. For simpler trail topologies, we propose theoretical models that consider both efficiency and accuracy criteria and present analytical results. For complicated graph topologies, we develop efficient heuristic algorithms with various heuristics. In addition, we analyze the difference of single hiker and multiple hiker scenarios with different hiking dynamics. After access point deployment is decided, the actual cost of search in individual search regions 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. Next, we present extensive experimental results to compare the performances of different methods and examine the accuracy of the mathematical models. A very fast heuristic method, divide-merge, is shown to outperform all others and finds near-optimal solutions. We also shows the effects of the graph topologies and number of access points on the solution qualities. Generally speaking, more access points lead to smaller search regions. Further improvement by moving the access points from vertices to edges is easily achieved when the number of access points is large or/and the average degree of vertices is small.

Finally, we extend our results by relaxing the assumption of the uniform distribution of the hiker missing probability. We analyze the problem complexity and present a general solution.

List of Tables

List of Figures

Contents

Chapter 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 [1, 2, 3, 4]. 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**onnectionless S**en**sor-Based Tracking System Using **Wit**nes**s**es) [1], which is proposed for search and rescue of subjects (e.g., people or wild animals) in emergency situations in wilderness areas such as the trail shown in Fig. 1.1. 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,

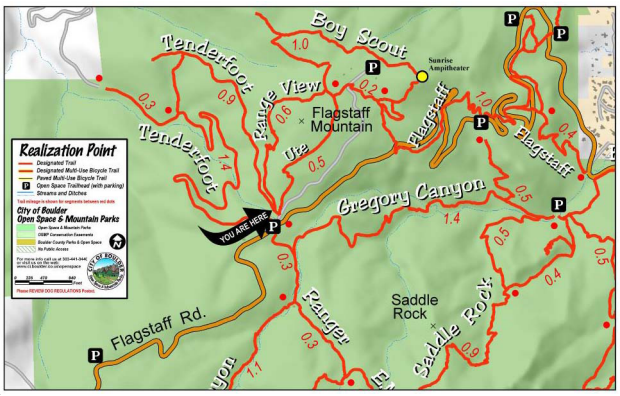


Figure 1.1: A map for trails outside Boulder, Colorado.

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 their 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 of 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 needed 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 makes it impractical to deploy a dense AP network.

In this thesis, we study several issues raised in the design of CenWits-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 search region containing the lost hiker; and (b) how to search efficiently in search regions with different topologies. The system is a multi-agent system with hikers as mobile agents. Together with APs, the hikers 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 thesis is organized as follows. In the next Chapter, we present an overview of the problem and our approach. In Chapter 3, we define a mathematical formulation of the problem with two objectives, search cost and location uncertainty, and discuss their trade-offs. We solve the AP placement problem of single-hiker model based on the location uncertainty objective in Chapter 4 and solve the problem in multi-hiker model in Chapter 5. We derive theoretical results on simpler trail topologies and present efficient heuristic algorithms for more complicated trail topologies. In Chapter 6, 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 Chapter 7, we present experimental results to compare the performances of different methods and to examine the accuracy of the mathematical models. We analyze the case with non-uniform hiker distribution and present the experimental result to support our analysis in Chapter 8. Finally, we discuss related work in Chapter 9 and

summarize the thesis in Chapter 10.

Chapter 2

Overview of Our Approach

A typical scenario of finding a lost hiker using a wireless sensor network system similar to CenWits 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. One or more search and rescue agents are sent to the region to find the lost hiker.

Our approach is similar yet different from the method proposed in CenWits. In CenWits, the lost region is determined based on the last-seen record of the lost hiker and his average hiking speed computed from earlier records. Several hot search regions are further identified according to the overlapping landscape (e.g. near the cliff). This is rather a simple approach and does not guarantee that the lost hiker will be found in the hot search regions. In contract, our approach places

the access points such that the whole trail map is partitioned into several disjoint subgraphs as trail segments. The last-seen record is thus used to identified the exact trail segment (TS) where the hiker is lost.

Our approach improve CenWits in terms of reducing the search cost. In CenWits, the access points are placed in arbitrary locations and used for witness records uploading only. The search region is determined based on the landscape and the expected travel distance of the lost hiker after he has been seen for the last time. The size of the search region highly depends on how soon the central rescue station retrieves the witness records. It can be fairly large if the hiker with the record approaches an AP very slowly or stop (e.g., camping) in the middle of no connection zone. Our approach solves the problem by means of partitioning the trail map into several smaller trail segment as described above. By reducing the size of each partition, we limit the upper bound of the search region's size. In addition, after a access point placement is achieved, an optimal search path plan is computed to minimize the travel time or search cost of the search and rescue agent(s).

Table 2.1 offers a comparison between CenWits and our approach. Fig. 2.1 shows the process of CenWits and Fig. 2.2 illustrates our approach.

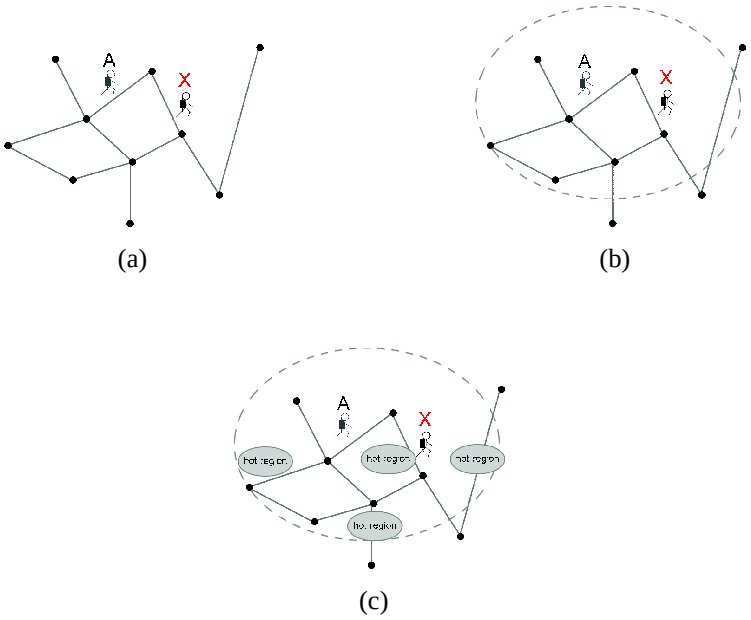
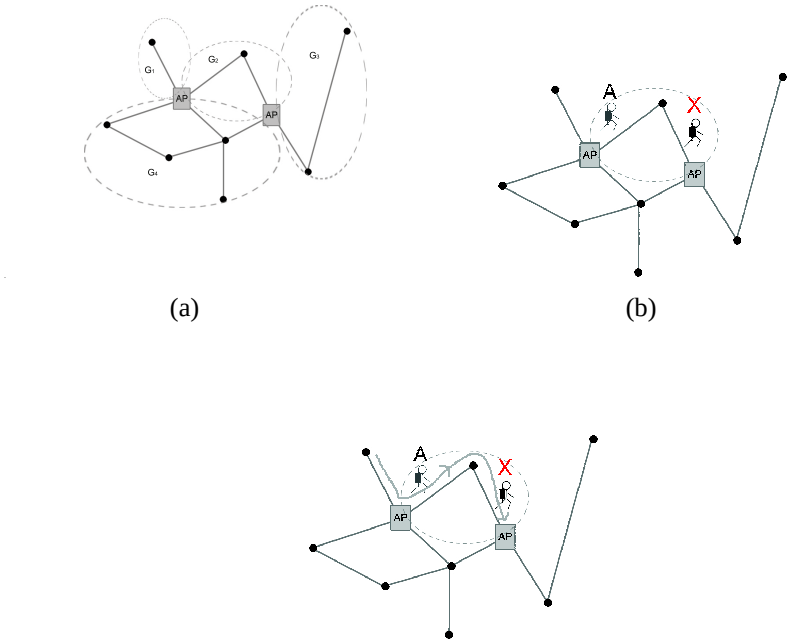


Figure 2.1: Illustration of CenWits system: a) a sample trail, the lost hiker is last seen at location A and is lost at location X; b) based on the average hiking speed and time elapsed since the hiker being seen at A, a possible lost region (the oval) is determined; and c) the region is mapped with the overlapping landscape and several hot search areas are identified.



(a) (b)

(c)

Figure 2.2: Illustration of our approach: a) two access points are placed and divide the trail into 4 subgraphs (as marked by the ovals); b) based on the last seen location A, the exact trail segment is identified as circled by the oval; and c) an optimal search path is planned (assume the rescue team starts from one trail end) with the smallest search cost.

2.1 Environment: Trails, Hikers, the Sensornet and APs

A trail map is represented as an undirected graph with edges for trail branches and vertices for intersections. We assume many hikers moving along the trails and switching trail branches only happens at intersections (vertices). In Chapter 4, we discuss the problem with only one hiker in the trail map and in Chapter 5 we present the case with many hikers walking in the trail.

Each hiker wears a sensor node and they form a loosely coupled sensornet with changing topologies. Only when two hikers with their sensor nodes come into a close range can a communication be established. The AP network, on the

Table 2.1: Comparison between CenWits and our approach

	CenWits	Our Approach
Localization method	1. Compute the possible range based on the last-seen location and average speed. 2. Use the overlapping landscape to estimate the hot search regions.	1. Partition the graph into smaller trail segments by placing several APs. 2. Identify the exact lost trail segment based on the last-seen location.
Search method	no	1. Optimal path planning based on trail segment topologies. 2. Implement efficient search path planning techniques according to the different capabilities of the search agents.

other hand, is static and fully connected. The sensornet exchanges information with the AP network when one hiker with a sensor node comes close enough to an AP and form a temporary connection.

In our research, we assume the APs has have knowledge of nearby hikers' heading and thus know the exact trail segments the hikers are moving into. This can be easily implemented by using simple inference technique to process the sensing data [5]. Furthermore, the methods developed can be easily extended to the case that the heading of the hikers is unknown to the APs.

2.2 Performance Measure

We focus on minimizing the expected search cost for all lost case. If the hiker gets lost in different trail segments, the expected search costs are different. A good solution for one single case may be a bad solution for another given case. Therefore, we assume the hikers can get lost at any parts of the trail map and try to minimize the expected search cost for all possible lost locations. In Chapter 3, we give a formal definition of the problem with two different objective functions.

We address the problem in two steps. First, we partition the trail map into many small trail segments using APs so that the expected search cost is minimized. Second, we perform an optimal search path planning process for every trail segment to minimize the actual search cost. The first step is addressed in Chapter 4 and 5 and the solutions for the second step are proposed in Chapter 6.

2.3 Achieving Smaller Trail Segments

To reduce the expected search cost, the first step is to achieve a partition with many small trail segments. For example, in Fig. 2.2(a), the two APs divide the trail map into 4 trail segments. Partitions G_1, G_2, G_3 are small and thus searching in those trail segments are efficient. However, the partition also results in a fairly large trail segment G_4, which require much higher searching cost. A better partition balances every trail segment so that the expected search cost when a hiker is lost in any possible trail segment is minimized. In Chapter 4, we address the problem of achieving better partition with only one hiker in the trail and in Chapter 5 we address the problem in the multi-hiker scenario.

2.4 Minimizing the Search Cost

In Fig. 2.2(c), an optimal search path is showed within one trail segment. One can also draw the search path for all other trail segments with the minimal travel cost. For a single ground rescue agent, the problem is identical to the classic Chinese Postman Problem [6]. In Chapter 6, we analyze the case with other types of search agents and also the scenarios where multiple search agents present.

Chapter 3

Problem Statement

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 denote the total weights of the graph, i.e., $l = \sum_{e \in E} w(e)$.

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* (TSs). 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. 3.1 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

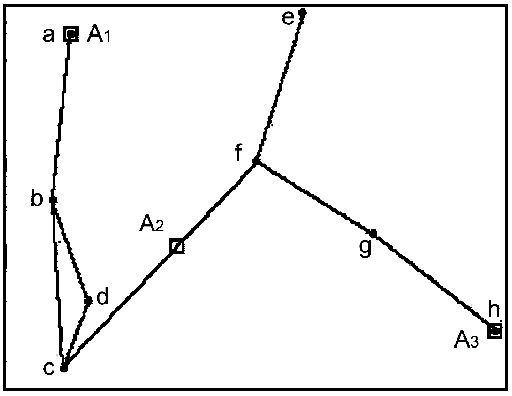


Figure 3.1: An example of AP placement in a trail graph. The three squares represent APs.

follows. Let $w_i, i \in [1, m_s]$, denote 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

$$p(G_i) = \frac{w_i}{l}. \tag{3.1}$$

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

$$C = \sum_{i=1}^{m_s} p(G_i)c(G_i) = \sum_{i=1}^{m_s} \frac{w_i \bullet c(G_i)}{l}. \tag{3.2}$$

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

$$\min C \text{ subject to } \sum_{i=1}^{m_s} w_i = l. \tag{3.3}$$

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 Chapter 6.

Solving Eq. (3.3) can only be done numerically and is computationally expensive because it is very time consuming to evaluate C for different AP locations. To make the problem more tractable, we use another metric, *location uncertainty*, in place of the real 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,

$$\sigma_i = \frac{1}{2\sqrt{3}} w_i. \tag{3.4}$$

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

$$U = \sqrt{3} E(\sigma_i). \tag{3.5}$$

Now, the new objective is to find an AP placement that minimizes U. Specifi-

cally, we solve the following problem:

$$\min U \text{ subject to } \sum_{i=1}^{m_s} w_i = l. \tag{3.6}$$

In our approach to 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. (3.6) 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. (3.3) 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 using U 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 Chapter 7.

Chapter 4

Access Point Placement in Single-Hiker Model

Based on the trail structure, we divide the problem into two categories: a simple edge or a graph.

4.1 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. The m APs (A_1, A_2, ..., A_m) are depolyed along the trail. In this case, with two APs deployed at the two trail heads, $m_s = m - 1$. From Eqs. (3.4) and (3.1), we have

$$U = \sqrt{3}E(\sigma_i) = \sqrt{3}\sum_{i=1}^{m_s} \sigma_i p(G_i) = \frac{1}{2l}\sum_{i=1}^{m_s} w_i^2. \tag{4.1}$$

Let $\overline{U}$ be the arithmetic mean of U. Define vector $\vec{w} = (w_1, w_2, \ldots, w_{m_s})$. If $w_i, i \in [1, m_s]$ is uniformly distributed in the interval $(0, l - w_1 - \cdots - w_{i-1})$, i.e., the access points are placed randomly along the trail, we can compute the probability density function $f(\vec{w})$ for vector $\vec{w}$, and with which we can compute $\overline{U}$,

$$\begin{aligned}\overline{U} &= \iint\cdots\int U f(\vec{w}) dw_1 dw_2 \cdots dw_{m_s-1} \\ &= \frac{\int\int\cdots\int U dw_1 dw_2 \cdots dw_{m_s-1}}{\int\int\cdots\int dw_1 dw_2 \cdots dw_{m_s-1}} \\ &= \frac{l}{m_s+1} \quad . \end{aligned} \tag{4.2}$$

The detailed deduction of the above formula is in Appendix A.

From Eq. (4.1) and by means of Lagrange function, we can solve the optimization problem in Eq. (3.6). 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.

4.2 Trails of a graph

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

$$\sigma_i = \frac{w_i}{2\sqrt{3}} = \frac{\sum_{e\in E_i} w(e)}{2\sqrt{3}},\ p(G_i) = \frac{w_i}{l} = \frac{\sum_{e\in E_i} w(e)}{l}. \tag{4.3}$$

The uncertainty U is

$$U = \frac{1}{2l}\sum_{i=1}^{m_s} w_i^2 = \frac{1}{2l}\sum_{i=1}^{m_s}(\sum_{e\in E_i} w(e))^2. \tag{4.4}$$

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

$$Var(F) = E(w_i^2) - (E(w_i))^2 = \frac{\sum w_i^2}{m_s} - (\frac{\sum w_i}{m_s})^2. \tag{4.5}$$

Thus, we can simplify U as

$$U = \frac{m_s}{2l}E(w_i^2) = \frac{m_s Var(F)}{2l} + \frac{l}{2m_s}. \tag{4.6}$$

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 an even placement may not result in an optimal solution. Generally speaking, a closed form solution does not exist for Eq. (4.6), 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, the APs can be placed anywhere along the trails, including the vertices and edges. Finding the optimal solution is very difficult. Instead, to find near optimal solutions efficiently, 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.

4.3 AP placement on vertices for graph trails

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. (4.4) is minimal. Enumerating all combinations requires $O(|V| \bullet C^m_{|V|})$ time. Here, we present four polynomial time 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) Group-exchanging Local Search. This heuristic approach tries to move a group of APs H at a time, which results in a much smoother curve and thus runs more iterations than method 1 (7.5 vs. 1.8 on average on our test cases). However, its time complexity is usually higher than method 1. Let b denote the size of H and $max(deg(V))$ denote the maximal degree of all vertices. The time complexity of method 2 is $O(|V|^2 \times b^{max(deg(V))})$.

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. It utilizes the existing algorithms for the graph partitioning problem which is studying how to achieve k balanced partitions such that 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 the 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 hypergraph $G^h = (V^h, E^h)$ where the vertices and edges are exchanged, and the weight of the vertices in G^h are the edge weight in G. Let $|E_c^h|$ denote the total number of edges connecting any two adjacent partitions. The problem becomes

$$\min |E_c^h| \text{ subject to } \forall i \in [1...m_s] \;\; w_i \leq \alpha \bullet \frac{l}{k}. \tag{4.7}$$

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 [7] 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. In every iteration, the algorithm selects a random partition 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 needed as to obtain a reasonable good solution, but $t \ll |V^h|$ ([8]). 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 with time complexity $O(|V|)$, 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.

4.4 Edge refinement - moving APs to edges for graph trails

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

$$W = \sum_{G_j \in \Omega} \sum_{e \in E_j} w(e). \tag{4.8}$$

Assume we can improve the solution by moving A_i to one of the adjacent trail segment G_j. Let G' denote the trail segment enclosed by the new A_i position and the original position. An example is shown in Fig. 4.1. Let $w_j = \sum_{e \in G_j} w(e)$,

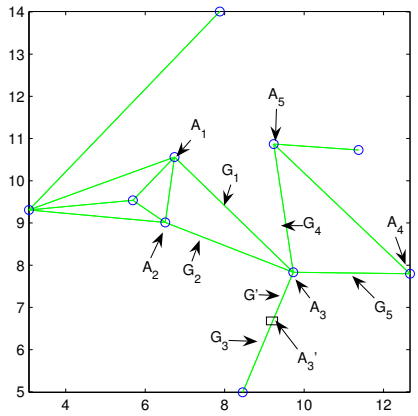


Figure 4.1: An example of moving an AP, A_3, from a vertex to an edge point, A_3'.

$w' = \sum_{e \in E'} w(e)$, and $h = |\Omega|$. 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 denote the two new trail segments with weights w^0 and w^1, respectively. The uncertainty of the new placement is

$$U' = \frac{1}{2l}(w^0 + w^1) = \frac{1}{2l}[(\sum_{x \in \{1,\ldots,h\}-\{j\}} w_x + w')^2 + (w_j - w')^2]. \tag{4.9}$$

whereas the uncertainty of the original placement is

$$U = \frac{1}{2l} \sum_{x \in \{1,\ldots,h\}} w_x^2. \tag{4.10}$$

The difference between the new placement and the original placement is

$$\Delta U = U' - U = \frac{1}{2l}(w'^2 + (W - 2w_j)w' + \sum_{x \neq y}^{x,y \in \{1,\ldots,h\}-\{j\}} w_x w_y). \quad (4.11)$$

Let $A = \sum_{x \neq y}^{x,y \in \{2 \ldots k\}-\{j\}} w_x w_y$. We have

$$\Delta U = U' - U = \frac{1}{2l}(w'^2 + (W - 2w_j)w' + A). \quad (4.12)$$

In order to make an improvement, ΔU has to be negative. We can derive a simple necessary (though not sufficient) condition to make an improvement, which is

$$w_j > \frac{W}{2}. \quad (4.13)$$

Therefore, in order to make an improvement by moving A_i, it has to have an adjacent TS whose weight exceeds half of the overall weight of all TSs adjacent to A_i.

For example, when all TSs have equal weight, we have

$$\Delta U = \frac{1}{2l}(w'^2 + (k-2)w_j w' + \binom{k}{2} w_j^2). \quad (4.14)$$

which is always positive. Therefore, it is impossible to make any improvement by moving A_i to one of the edges.

Taking the derivative of Eq. (4.11) and setting it to 0, we can compute the w'

value for the minimal uncertainty (if it exists) as follows

$$w' = w_j - \frac{W}{2} \quad \text{for} \quad w_j \geq \frac{W}{2}. \tag{4.15}$$

There are two important properties of the new placement found by Eq. (4.15): the new placement is both half-divided and unique. They are important in proving the correctness of the algorithm.

Lemma 4.4.1 *In order to find an optimal solution by Eq. (4.15), the original graph has to be 1-edge-connected and the new placement is on the edge whose removal disconnects the graph.*

Proof: Assume the original graph is not 1-edge-connected. In such a case, removing the edge at which the new placement is does not disconnect the graph. Therefore, the new placement results in a bigger partition, which is contradicted to the assumption that the new placement makes an improvement.

Theorem 4.4.2 *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. (4.15) divides G^- into two equal graphs G_1^- and G_2^- with $w(G_1^-) = w(G_2^-)$, where $w(G_x)$ is the sum of all edge weights of graph G_x (e.g., $w(G^-) = W$). We say that v'* half-divides *the graph G^-.*

Proof: As we know from lemma 4.4.1, removing the edge where v' is disconnects G^- and thus we only have two new partitions G_1^-, G_2^- after we make the

change. We have

$$w(G_1^-) = w_j - (w_j - \frac{W}{2}) = \frac{W}{2}$$
$$w(G_2^-) = W - w(G_1^-) = \frac{W}{2}.$$

Therefore, we have $w(G_1^-) = w(G_2^-)$.

Theorem 4.4.3 *There is at most one new placement found by Eq. (4.15). The solution is* unique.

Proof Assume there is a placement v_1' that divides G' into two subgraph G_1', G_2' with $w(G_1') = w(G_2')$, and there is another distinct placement v_2' found by Eq. (4.15). According to lemma 4.4.1, v_1', v_2' partition G' into three disconnected subgraph G_1, G_2, G_3 and assume G_2 to be the graph between v_1', v_2'. We have

$$\begin{aligned} w(G') &= w(G_1) + w(G_2) + w(G_3) \\ &= \frac{w(G')}{2} + w(G_2) + \frac{w(G')}{2} \\ &= w(G') + w(G_2) \\ &> w(G'). \end{aligned}$$

which shows a contradiction.

According to theorem 4.4.2, the new location can be determined by examining

whether there is a place that half-divides the trail segment. In addition, theorem 4.4.3 guarantees the new placement we found is unique and local-optimal.

A greedy algorithm is developed based on the theorems as follows:

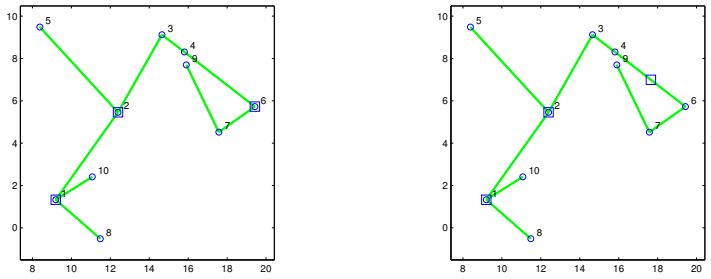
1. Start with an initial placement.

2. In 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 marginal after several iterations. Thus we set the maximal number of iterations to be relatively small, e.g., 10.

Fig. 4.2(a) and 4.2(b) shows an sample placement found by the Divide-merge algorithm with 3 AP and the improvement made by edge refinement algorithm. Between two candidate vertices to move, vertex 1 and vertex 6, only the change of vertex 6 will result in a smaller uncertainty, which is a vertex with degree 2. Similarly, during our experiment, we have observed that most improvements were made with the vertices whose degrees were less than 3. This is because under such condition, the adjacent trail segment only need to satisfy Eq. (4.13), whereas when the degree is larger 3, they also have to satisfy that Eq. (4.11) < 0, which is a much more strict condition.



(a) (b)

Figure 4.2: A sample trail with 10 vertices and 3 AP. (a) A placement by the divide-merge algorithm, the square signs are the AP placement and (b) The placement reached by the edge refinement algorithm

Chapter 5

Access Point Placement in Multi-hiker model

When there are multiple hikers coming across each other, they provide more location information of each other. In the scenario of multiple hikers, we mainly study two hiker movement modes: one-way and round-trip. Let x be the number of other hikers that the hiker h runs into during a hike.

5.1 One-way single path

First, consider a trail that only has a single path along which the hiker moves in one direction. If all hikers start from the same trail head at the same time and have the same speed, i.e., hiking as a group, then the location of the lost one can be precisely determined. If all hikers start at different times and have the same

speed, then they will not witness each other. This is equivalent to the single-hiker scenario.

A more interesting case is that we assume the hiker h is moving one-way on the path from one trail end to the other while all other hikers move one-way, at different speeds, in either direction. The hiker h's location is recorded each time h comes into the range of another hiker or an access point. So h can have as many as $m + x$ recorded locations, which partition the trail into $m_s + x$ segments ($m_s = m - 1$). Let $S_i, i \in [1, m + x]$ denote these recorded locations, and w_i' denote the distance from S_i to S_{i+1}. If h is lost between S_i and S_{i+1}, the location S_{i+1}, at which h is supposed to run into another hiker or access point, will not be recorded. If S_{i+1} is a location of a hiker, say h_2, then S_{i+1} can be figured out from the hiking speeds, witness time and locations of the both hikers. On the other hand, if the nearest witness is an access point, then S_{i+1} is the position of that access point.

Let Y_i' be the location between S_i and S_{i+1} at which h is lost, and σ_i' be the standard derivation of Y_i'. Thus,

$$\sigma_i' = \frac{1}{2\sqrt{3}} w_i', i \in [1, m_s + x]. \tag{5.1}$$

Let U be the location uncertainty of the lost hiker h,

$$U = \sqrt{3} \sum_{i=1}^{m_s+x} \sigma_i' p(\sigma_i') = \frac{1}{2l} \sum_{i=1}^{m_s+x} w_i'^2. \tag{5.2}$$

Minimal U is reached when $w'_i = w'_j, i, j \in [1, m_s + x]$,

$$U_{min} = \frac{l}{2(m_s + x)}. \tag{5.3}$$

It means that we should place more access points where people are sparse, and less where people are dense. On the other hand, uniform distribution of hikers leads to uniform distribution of access points.

In practice, it is hard to satisfy the condition $w'_i = w'_j, i, j \in [1, m_s + x]$, since the motion of hikers is beyond control. Therefore, the mean location uncertainty $\overline{U}$ is more practical than U_{min}. Suppose $w'_i, i \in [1, m_s+x]$ is uniformly distributed in the interval $(0, l-w'_1-\cdots-w'_{i-1})$. In the same way as presented in single-hiker model, we can get

$$\overline{U} = \frac{l}{m_s + x + 1} \approx \frac{l}{m_s + x}. \tag{5.4}$$

5.2 Round-trip single path

In a round-trip single path case, every hiker turns back at an arbitrary location and return to the starting point. For simplicity, we assume they do not turn back until they reach the other end of the trail. The hiker h runs into each of the access points or hikers twice at most, which means at most $2(m + x) - 1$ pieces of recorded locations. These locations partition the trail into $2(m_s+x), m_s = m-1$ segments.

Thus the location uncertainty is

$$U = \sqrt{3} \sum_{i=1}^{2(m_s+x)} \sigma_i' p(\sigma_i') = \frac{1}{4l} \sum_{i=1}^{2(m_s+x)} w_i'^2. \tag{5.5}$$

which, as above, leads to

$$U_{min} = \frac{l}{2(m_s + x)}. \tag{5.6}$$

$$\overline{U} = \frac{2l}{2(m_s + x) + 1} = \frac{l}{m_s + x + \frac{1}{2}} \approx \frac{l}{m_s + x}. \tag{5.7}$$

5.3 Trails with graph structures

We start with a graph with access points placed on each node $v \in V$ and on some long edges $e \in E$. Still, consider the graphs with one-way edges first. For edge $e \in U$, let $x(e)$ denote the number of hikers encountered on edge e; and $w_i'(e)$ denote the distance between the i^{th} and $(i+1)^{th}$ recorded location of h on edge e. Then

$$U = \sqrt{3} \sum_{e \in E} \sum_{i=1}^{m_s(e)+x(e)} \sigma_i'(e) p[\sigma_i'(e)] = \frac{1}{2l} \sum_{e \in E} \sum_{i=1}^{m_s(e)+x(e)} w_i'(e)^2. \tag{5.8}$$

where $m_s = \sum_{e \in E} m_s(e), x = \sum_{e \in E} x(e)$.

It can be inferred that U_{min} is the same as in Eq. (5.3). Similarly, for graphs with round-trip edges, U_{min} has the same format as before.

Eq. (5.4) applies to graph with one-way edges. If $m_s + x \gg 1$, we can rewrite

$\overline{U}$ as,

$$\overline{U} \approx \frac{l}{m_s + x}. \tag{5.9}$$

Based on the derivation in the previous section, it can be deduced that Eq. (5.9) applies to graphs with round-trip edges, where hikers are supposed to return when they reach the other end of the trail.

Therefore, to minimize the location uncertainty on a graph in the scenario of multi-hikers, access points should be placed to supplement the hiker distribution, i.e., more access points should be placed where people are sparse, less where people are dense, and uniform where people's distribution is even. Given the expected number of hikers on each trail segment, we use Eq. (5.9) 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 $\overline{U}$.

Chapter 6

Minimizing Search and Rescue Cost

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 us take a look at 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 the trail segment.

6.1 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:

$$c^M(G_i) = c'(P) + \sum_{e \in E_i} n(e)c(e). \tag{6.1}$$

where E_i is the set of all edges in G_i, $n(e)$ is the number of 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) [6] on an undirected graph. There exist polynomial time algorithms to find the optimal solution. Let w_i denote 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).

A general approach for solving CPP is to construct an Euler tour from the graph by adding edges if necessary. This is based on the fact that an Euler path

has only two vertices with odd degrees at the end points and all other vertices even degrees. The basic steps of constructing a CPP path are

1. Identify the N vertices with odd degrees. N is always even.

2. Find a *minimum-length pairwise* matching of the N vertices and find the $N/2$ shortest path between the two vertices of each pair.

3. Add the edges to the graph.

4. Find an Euler tour, e.g., with classic Fleury's algorithm.

The *minimum-length pairwise* operation finds a set of pairs that covers all N vertices such that the sum of all the shortest paths between the two vertices of each pair is minimal. It is the most costly operation in the process, but an $O(N^3)$ algorithm is available [9].

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 $l_t, (t = 1, ..., n)$ denote the n tour segments. Let $r_j, (j = 1, ..., (n-1))$ denote 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 $w(l_t)$ denote the weight of tour segment

l_t and $w(r_j)$ denote the weight of redundant segment r_j, the expected search cost when the lost hiker is found in tour segment l_t is

$$c^E(l_t) = c'(P) + W^{b(l_t)} + \frac{w(l_t)}{2}. \tag{6.2}$$

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

$$p(l_t) = \frac{w(l_t)}{w_i}. \tag{6.3}$$

Therefore, the expected search cost in G_i is

$$\begin{aligned} c^E(G_i) &= \sum_{t=1}^{n} p(l_t) c^E(l_t) && (6.4)\\ &= \sum_{t=1}^{n} \frac{w(l_t)}{w_i} [c'(P) + W^{b(l_t)} + \frac{w(l_t)}{2}] && (6.5)\\ &= c'(P) + 0.5w_i + \sum_{t=1}^{n} \frac{w(l_t)}{w_i} [\sum_{j=2}^{t} w(r_{j-1})]. && (6.6) \end{aligned}$$

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

To simplify Eq. (6.4), we have

$$\begin{aligned}
& \sum_{k=1}^{n} \frac{w(l_t)}{w_i}[W^{b(l_t)} + 0.5w(l_t)] \\
= & \sum_{k=1}^{n} \frac{w(l_t)}{w_i}[\sum_{j=2}^{t}(w(l_{j-1}) + w(r_{j-1})) + 0.5w(l_t)] \\
= & \sum_{t=1}^{n} \frac{w(l_t)}{w_i}[\sum_{j=2}^{t} w(l_{j-1}) + 0.5w(l_t)] \\
& + \sum_{t=1}^{n} \frac{w(l_t)}{w_i}[\sum_{j=2}^{t} w(r_{j-1})] \\
= & \ 0.5w_i + \sum_{t=1}^{n} \frac{w(l_t)}{w_i}[\sum_{j=2}^{t} w(r_{j-1})].
\end{aligned}$$

Thus Eq. (6.4) can be simplified as

$$c^E(G_i) = c'(P) + 0.5w_i + \sum_{t=1}^{n} \frac{w(l_t)}{w_i}[\sum_{j=2}^{t} w(r_{j-1})]. \tag{6.7}$$

A heuristic method to reduce the expected search cost is to perform a greedy walk on the Euler graph found by the CPP algorithm, always choosing unvisited edges when possible. This is based on the observation from Eq. (6.7) 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 results show that the heuristic method outperforms the classic Fleury's algorithm with respect to achieving

lower expected search cost.

6.2 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

$$c^M(G_i) = c'(P) + \max_{x=[1...k]} \left(\sum_{e \in E_i^{T(x)}} n(e)w(e) \right). \tag{6.8}$$

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 [10] that utilizes the principle of tabu search.

The expected search cost for M-GSA is

$$c^E(G_i) = c'(P) + \frac{w_i}{2} + \max_{x=[1,...,k]} \left(\sum_{t=1}^{n^x} \frac{w(l_t^x)}{w_i} [\sum_{j=2}^{t} w(r_{j-1}^x)] \right). \tag{6.9}$$

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

$$c^M(G_i) = c'(P) + w(L_{i,u}). \tag{6.10}$$

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

$$c^E(G_i) = c'(P) + 0.5w(L_{i,u}). \tag{6.11}$$

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).

6.3 Single air SaR agent (S-ASA)

The problem of minimizing the search cost of an S-ASA is quite similar to that of a single ground agent except that an 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 the 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

$$c^M(G_i) = c'(P) + \sum_{e \in E_i^T} n(e)w(e). \tag{6.12}$$

where E_i^T is the set of all edges (include dummy edges) traveled. Eq. (6.12) can be computed using the existing algorithms, such as the one based on local search in [11].

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

$$c^E(G_i) = c'(P) + \frac{w_i}{2} + \sum_{t=1}^{n} \frac{w(l_t)}{w_i} [\sum_{j=2}^{t} w(r_{j-1})]. \tag{6.13}$$

which is similar to Eq. (6.7). The method proposed for S-GSA can also be applied here.

6.4 Multiple air SaR agents (M-ASA)

Performing the same graph transformation on G_i as in the single air SaR agent case, 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. (6.8) and Eq. (6.9). 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.

6.5 Consider the entering point of the trail segment

In the case of S-GSA, if P in Eq. (6.1) is not fixed, finding a better entering point to the trail segment helps in reducing the internal travel distance. For example, if G_i is an Euler path and we start from one end point, the overhead of the redundant edge is 0. If we start at the middle of the Euler path, on contrary, the total length of edges travelled more than once is up to $\frac{w_i}{2}$. However, changing the starting point may result in a higher $c'(P)$ value and finding the minimal value for all c^M and c^E becomes the new objective. A simple approach is testing all the possible starting vertices. Because there exists a polynomial algorithm for CPP, the new algorithm is still polynomial.

Among all other SaR scenarios, how to choose the starting vertex also affects the overall travel time. If we adopt the same technique as in S-GSA, the complexity increased $O(|V^{G_i}|)$ times where V^{G_i} is the set of vertices of the trail segment C. Given we chooses a polynomial algorithm for the original problem, the new optimization problem is still solvable in polynomial time.

In the rest of the thesis we omit the $c'(P)$ for both simplicity and the fact $c'(P) << \sum_{e \in E^{G_i}} c(e)$ in most cases. For example, one common search scenario

involves transporting several SaR teams with K9 search dogs by trucks that travel at a speed of more than 20 mph on hiking trails, whereas the search process by the K9 teams advances less than one mile an hour on average. Under such a scenario, the cost of transporting the search teams to the trail segment is neglectable due to its marginal effects on the final search cost.

Chapter 7

Experimental Results

We implemented the test environment in MATLAB. First, a random map generator created random trail maps by placing n vertices randomly inside a square region of size $5n \times 5n$. Nearby vertices were first connected by edges with the lengths uniformly distributed in interval $(0, 5)$. When two edges crossed over, one was selected to be removed probabilistically. The result was a connected planar graph with the maximal node degree no more than D which was usually set to 5 unless otherwise specified.

Next, a partition was achieved using either of the following AP placement algorithms with varying number of access points.

1. Divide-merge algorithm (DM).

2. Local search algorithm that only moves one AP at every iteration (LS).

3. Group-exchanging local search algorithm that moves m^- APs at every iter-

ation (GELS), $m^- = max(5, m)$ where m is the number of access points placed.

4. Optimal solution of placing AP on vertices only (optimal). The solution is achieved by enumerating all possible placements.

5. Hyper-edge partitioning algorithm (HKP). An extra phase of hyper-edge refinement [7] is performed when needed.

6. Divide-merge algorithm with edge refinement (DMR). A limit of the number of edge refining iterations d is set to 1,10 or 20.

The first five algorithms restrict the placement of APs to vertices, whereas the last one can place APs on edges.

After we achieved an AP placement, for each trail segment, we performed a Chinese Postman algorithm to construct an Euler path with the minimal search cost in the worst case c^M, and used the greedy heuristic method proposed in Chapter 6 to find the exact search path that minimize the expected search cost c^E. We also ran the classic Fleury's algorithm in finding the exact search path as a comparison to the greedy method.

Finally, we randomly generated 30 lost locations for each trail map and got the average search cost.

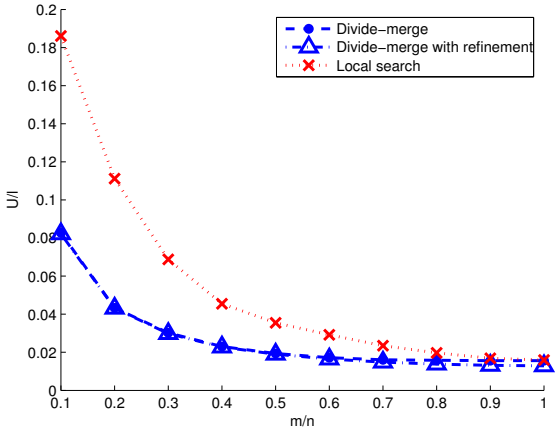


Figure 7.1: Comparison of local search and divide-merge (DM) algorithms for placing AP on vertex and DM with edge refinement that allows APs to be placed on edges. x is the percentage of access point number m out of n. y is the normalized uncertainty with $y = U/l$ where l is the total length of the trail map.

7.1 Compare Divide-merge (DM) with Local Search (LS)

First, we compare the results of five algorithms: divide-merge (DM), local search (LS), and the divide-merge with refinement (DMR). Number of vertices n is randomly drawn from the vector [10,20,...100] and the vertices are placed in a square field of size $5n \times 5n$. The number of access point $m = \beta n$ where $\beta = 0.1, 0.2, ..., 1$. Fig. 7.1 shows the performance of the three algorithms which is normalized to the ratio of the uncertainty U to the total trail length l because of the varying problem size. Each data point corresponds to the average value from 100 runs.

We have observed the following facts from Fig. 7.1. First, divide-merge al-

gorithm generates consistently better result than the local search with less than 50% uncertainty on average when m is less than 50% of n. We also notice that the DM algorithm ran much faster than the LS algorithm, very fast even when the number of vertices is large. In addition, the data of DM 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 vertices with degree 1. Apparently, deploying an AP on a degree 1 vertex will not reduce the uncertainty because it is already the boundary of a trail segment. 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 continues to improve after $m = 0.7n$.

Fig. 7.2 further shows the difference of DM and LS algorithms in the large size problem with 100 vertices in a field of size 500×500. We tested the problem with $m = 1, 2, ..., 100$ access points and each data point represents the average uncertainty of 30 runs. The DM algorithm again is shown to outperform the LS algorithm with dramatic improvement when m is small ($m < 10$).

7.2 Compare DM, DMR and LS algorithm to the optimal and GELS

In the next set of experiments, we test the algorithms of DM, DMR and LS and compare them with the optimal solution and GELS in smaller size problems of 10 vertices in a field of size 50×50. The number of access points is set to $1, 2, ..., 10$.

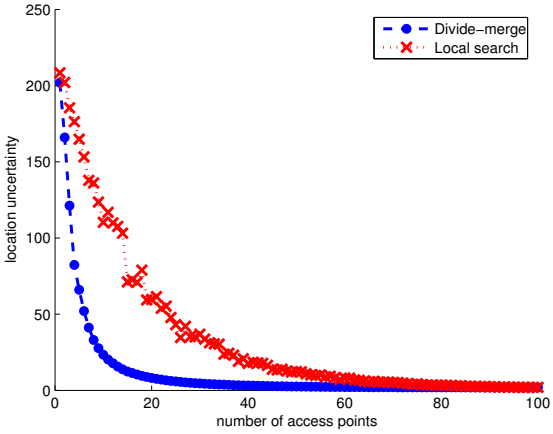


Figure 7.2: Comparison of local search and divide-merge (DM) algorithms at large trails with 100 vertices.

The test is not feasible in large size problems because of the running time of optimal and GELS algorithms.

Fig. 7.3 shows the location uncertainty values of the results found by the five algorithms for different number of access points. Each data point is computed as the average of 30 runs. LS algorithm produced the worst result, while the others performed quite similarly. Between the two local search algorithms, GELS achieves better solution by producing a smoother search landscape that allows more hill-climbing iterations (7.5 iterations of GELS vs. 1.8 iterations of LS on average). In addition, the solutions of DM and GELS are very close to the optimal solutions, and DM is slightly better. However, DM runs much faster than GELS, and is proved to be scalable in the first set of experiments.

The benefit of edge-refinement in DMR is significant when the number of APs is large, as shown in Figs. 7.3 and 7.4. In Fig. 7.4, the solutions of DM and DMR

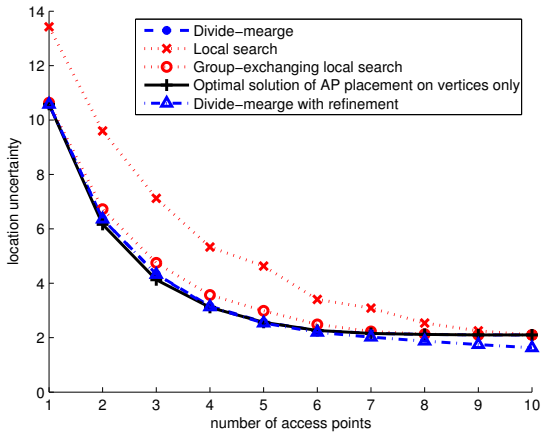


Figure 7.3: Comparison of four methods for placing AP on vertex and DM with edge refinement that allows APs being placed on edges.

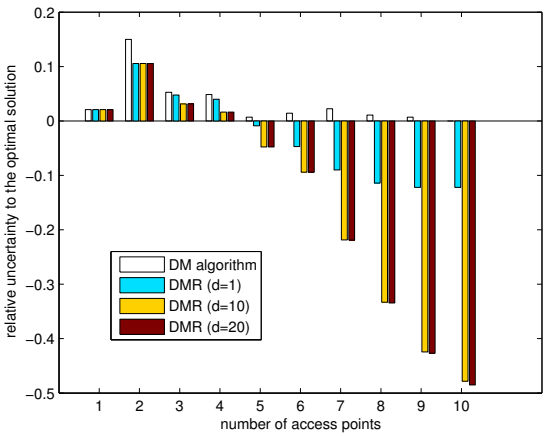


Figure 7.4: Comparison of DM without and with edge refinement of different iterations. $b = 5$ in group-exchanging local search.

with different iterations (1, 10, and 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 iteration, 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.

We further demonstrate the effects of the graph topologies on the performance of DMR algorithm in Fig. 7.5. Number of vertices n was randomly drawn from the vector [10,20,...100]. The maximal degree $max(deg)$ of any vertices is randomly generate within the range of [3 9] and the exact degrees of all vertices are uniformly distributed within [1 $max(deg)$]. The average degree of the trail map is thus approximately $\frac{1+max(deg)}{2}$.

We have observed that DMR performs better in the trails with smaller average degrees. Actually, during our experiment, most improvements were made with the vertices whose degrees are less than 3, which is easier to be satisfied when most vertices in the trail map have small degree.

7.3 Compare Divide-merge with Hyper-edge Partitioning Algorithm

Next, 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 [7], which is one of the two widely

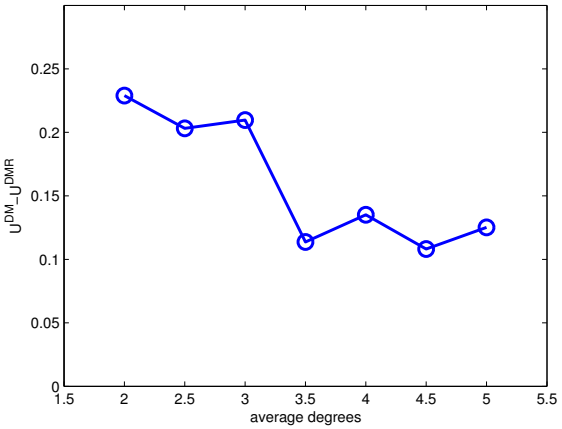


Figure 7.5: Compare the improvement made by DMR v.s. the average degrees of all vertices. y is reduction in uncertainty by DMR $\Delta U = U^{DMR} - U^{DM}$.

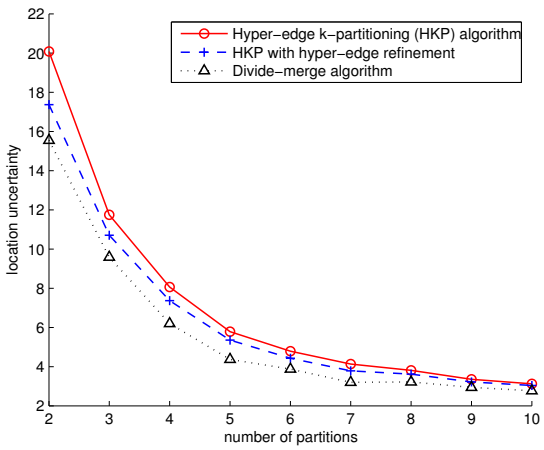


Figure 7.6: Comparison of DM, HKP, and HKP with hyper edge refinement.

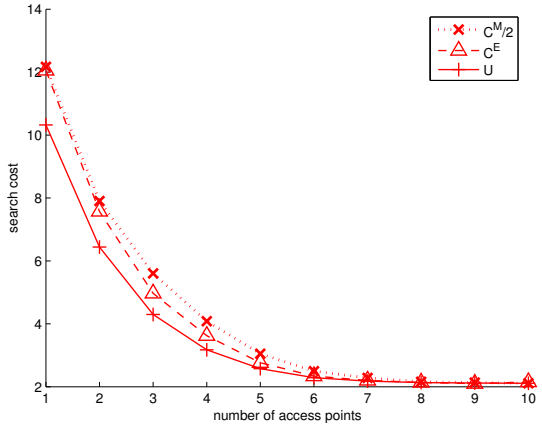


Figure 7.7: Comparison of values of three difference metrics, U, $c^M/2$, and c^E, on the solutions of DM.

used refinement methods in hypergraph partitioning problem (The other one is the FM algorithm [12]). We choose HER for its simplicity and good performance.

Random trail graphs in square regions with size 150×150 and 30 vertices were used in the experiments. The lengths of the edges are still in interval $(0, 5)$. 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. 7.3 shows that DM is consistently better than HKP+HER, which in turns is better than HKP.

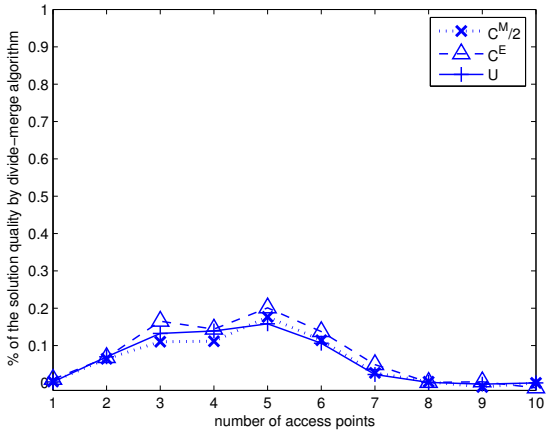


Figure 7.8: The differences of U, c^M, and c^E values between GELS and DM normalized over DM solutions, respectively.

7.4 Comparison of Different Objective Functions

Finally, we show that the objective of location uncertainty U is a good approximation to the real search cost C as in Eq. (3.2). In a field of size 50×50 with 10 randomly generated vertices, using the results of the divide-merge (DM) algorithm and the search path planning-testing process mentioned at the beginning of this section, Fig. 7.7 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 group-exchanging local search

(GELS), 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. 7.8. The y-axis is the normalized difference between the GELS results and the corresponding DM results:

$$y = \frac{Q_{MMLS} - Q_{DM}}{Q_{DM}}. \tag{7.1}$$

where Q_{MMLS} denotes the average value of GELS 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.

We also compare our greedy heuristic algorithm in finding the exact search cost to the classic Fluery's algorithm. The actually cost of the solution found by the greedy algorithm is actually c^E in Fig. 7.7 and Fig. 7.8 and the cost of the solution found by Fluery's algorithm is approximately $c^M/2$. The greedy algorithm achieves smaller expected search cost when the number of access points is between 20% and 50% of the number of vertices.

Chapter 8

The Effect of Non-uniform Hiker Missing Probability

8.1 Distribution on a Round-trip Scenario

So far we only assume that the hiker missing probability is uniform along the trail. However, the probability may follow different distribution. Consider a round-trip scenario on a single path where a hiker will turn back at a random position and the distribution of the turning points is uniform along the trail. The probability of a hiker being lost at a given point decrease linearly along the trail. Assume the probability at the trail head H is p_0. Let x_i denotes the distance from access points A_i to the trail head. The probability at A_i is $\frac{p_0(l-x_i)}{l}$. Then the probability in the trail segment G_i between A_i and A_{i+1} is

$$
\begin{aligned}
p(G_i) &= \int_{x_i}^{x_{i+1}} \frac{p_0(l-a)}{l} da \\
&= \frac{p_0}{l}(la - \frac{a^2}{2})|_{x_i}^{x_{i+1}} \\
&= \frac{p_0}{l}[l(x_{i+1} - x_i) - \frac{x_{i+1}^2 - x_i^2}{2})].
\end{aligned}
$$

When $x_{i+1} = l$ and $x_i = 0$, the value of $p(G_i)$ is 1. We can then calculate the value of p_0 as

$$
p_0 = \frac{2}{l}, l > 2. \tag{8.1}
$$

and the value for $p(\sigma_i)$ as

$$
p(G_i) = \frac{(x_{i+1} - x_i)(2l - x_{i+1} - x_i)}{l^2} = \frac{w_i(2l - w_i - 2x_i)}{l^2}, i \in [1, m_s]. \tag{8.2}
$$

From Eq. (8.2), we have

$$
\begin{aligned}
U &= \sqrt{3} \sum_{i=1}^{m_s - 1} \sigma_i p(G_i) \\
&= \frac{1}{2l^2} \sum_{i=1}^{m_s - 1} w_i^2 (2l - w_i - 2x_i) \\
&= \frac{1}{2l^2} \sum_{i=1}^{m_s - 1} w_i^2 (-2 \sum_{j=1}^{i-1} w_j - w_i + 2l).
\end{aligned}
$$

When $m_s = 3, w_1 + w_2 = l$ we have

$$U = (\frac{1}{2} - \frac{(w_1 - w_1^2)(3l - 2w_1)}{2l^3})l. \tag{8.3}$$

Get the derivative of Eq. (8.3) and set it to 0. We then have the optimal solution when

$$w_1 = \frac{5 - \sqrt{7}}{6} \bullet l \approx 0.39l. \tag{8.4}$$

which is slightly smaller than half of l.

When $m_s > 3$, we introduce a new variable λ and derive the Lagrange function as

$$\frac{\partial}{\partial w_i}(U - \lambda(\sum_{i=1}^{m_s-1} w_i - l)) = 0. \tag{8.5}$$

which can be derived to

$$4w_i(l - \sum_{j=1}^{i-1} w_j) - 3w_i^2 - 2\sum_{k=i+1}^{m_s-1} w_k^2 = 2\lambda l. \tag{8.6}$$

The equation does not have a closed-form solution. However, we can use numerical optimization to find the local minimal near the evenly placement. In the numerical solutions, the more likely a hiker will be lost in a segment, the more access points should be placed in it. A sample of such placement is shown in Fig. (8.1);

In Fig (8.2), we compare the uncertainty for the even placement and the placement found by our approach. The figure shows a slight improvement (about 10%).

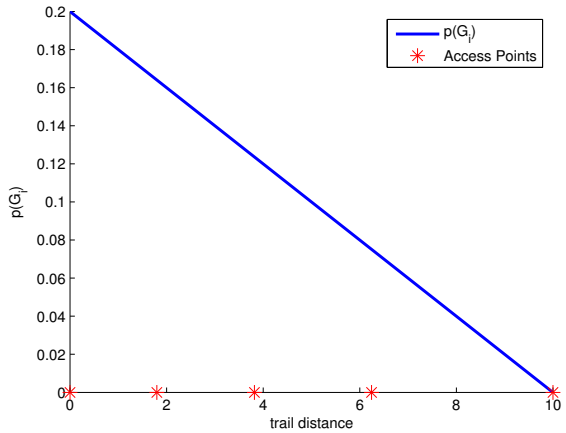


Figure 8.1: a sample solution with 5 access points in the scenario where the probability of a hiker being lost is linear decreasing along the trail

This is probably because we started from the even placement and the local minimal near it did not have a significant improvement. The uncertainty can be further reduced (though slightly) if we started from a random point. However, it can also get no improvement at all. Therefore, we think starting from the even placement is a good approach to follow.

8.2 General Solutions

We apply the above method to solve the scenario where the probability of a hiker being lost follows an arbitrary distribution. Similarly, a general closed-formed solution does not exist. Furthermore, numerical methods are not efficient. During our experiment with MATLAB solvers, we observed that in a finite number of iterations, it may not be able to find a better solution than the evenly distribution,

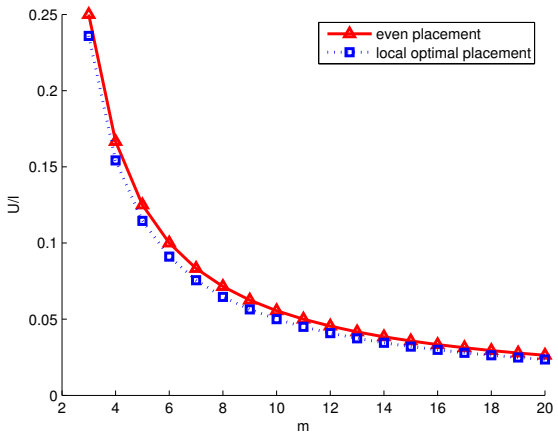


Figure 8.2: solution quality of the evenly placement v.s. the optimal placement in the scenario where the probability of a hiker being lost is linear decreasing along the trail

and the solver is rather slow and is only able to solve very small problems, e.g., $m_s = 3$.

Chapter 9

Related Work

9.1 Access point placement

The AP placement problem is essentially a graph partitioning problem, which 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 algorithms is called the spectral partitioning [13, 14, 15] which is quite expensive. Another class uses the geometric information of the graph (if the coordinates of the vertices are known) [16, 17, 18]. The geometric methods usually run faster than the spectral method, but the solution is usually worse. The third class of algorithms uses 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 to as uncoarsen phase when a refinement is usually adopted [8, 19, 20, 21, 22]. The multilevel approach usually provides better solution than spectral methods at lower cost [23].

Our problem is also related to graph-connectivity problems of determining the minimal number of vertices (or edges) needed to remove in order to partition the original graph into two or more disconnected graphs. They are also known as the problems of computing the edge-connectivity and vertex-connectivity. Many algorithms have been developed over the years. Most of them compute the connectivity by solving a number of max-flow problems [24] [25] [26] [27]. The solutions to graph-connectivity problems can be used as the upper- and lower-bound of number of the necessary access points for our problem.

9.2 Search and rescue path planning.

The problem can be formulated as the Chinese Postman Problem (CPP) on an undirected graph [6]. 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 [9].

Another problem related to CPP is Rural Postman Problem (RPP) [28], 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 [28]. Recently some heuristic methods based on either local search or Monte Carlo principles are proposed for RPP [11, 29, 30, 31, 32].

The problem of multiple SaR agents can be transformed to the k-Chinese Postman Problem (k-CPP) [33]. Instead of a single postman, a group of postmen are assigned the job of traveling all graph edges efficiently. Although a polynomial time optimal algorithm exists for the objective of minimizing the overall traveling distance or time for all postmen, an alternative objective that tries to minimize the maximum travel distance of any single postman makes more sense in the search and rescue operation. This is known as Min-Max k-CPP and is NP-hard. A heuristic method with $(2 - 1/k)$ approximation has been developed [33].

In recent years, there were many works that proposes new variances of either CPP or RPP and they were all proven to be NP-hard. One variance of the k-CPP problem was proposed by Degenhardt in 2004. It is called Minimum Absolute Deviation (MAD) k-CPP that tries to balance the workload of postmen by minimize the sum of all deviations from a fix travel time/distance for a single postman [34]. In 1999, Garfinkel and Webb introduced a variance of RPP the Crossing Postman Problem (XPP) [35]. It differs from RPP in a way that a postman is allowed to leave an edge and cross to another edge at a point other than the original vertices. It is quite similar to the case where an air SaR agent ceases the search in the middle of a trail and cross over the un-walking zone (e.g. cliff and lake) and fly to another trail. Another variance of RPP was described by Araoz et al. [36] in 2006 as Privatized Rural Postman Problem (P-RPP). A P-RPP modifies a

classic RPP by adding a cost $c(e)$ for every edge e and a gain $b(e)$ that only has a non-zero value when e is first visited. It also assumes the graph has a vertex as a central depot d. The objective is to find a cycle C, not necessarily simple, so that $f(C) = \sum_{e \in C} [b(e) - t(e)c(e)]$ is maximized and the depot d is passed through at least once, where $t(e)$ is the time the postman visit edge e. Therefore, instead of requiring travelling every edges, a P-RPP tries to maximize the profit ($gain - cost$). It is originated from another similar work [37] that modifies the classic Travelling Salesman Problem (TSP) such that although not all vertices are necessarily visited once, the profit of the salesman is maximize.

Chapter 10

Conclusion

In this thesis, 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 expected or maximal search cost. In simulation, we compare different algorithms and show that the solution quality obtained by an efficient heuristic method, divide-merge, is very close to the optimal solution.

In the future work, we plan to study the case of unknown hiking direction and multiple hikers in the optimal AP placement problem. Furthermore, we plan to study the SaR operations where more than one type of SaR agents are 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).

APPENDIX

Detailed deduction of Eq. (4.2)

Eq. (4.2) is used to model the relationship between the localization error and the number of access points. It is represented as follows,

$$\begin{aligned}
\overline{U} &= \frac{\int_0^l dw_1 \int_0^{l-w_1} dw_2 \cdots \int_0^{l-w_1-\cdots-w_{ms-2}} U dw_{m_s-1}}{\int_0^l dw_1 \int_0^{l-w_1} dw_2 \cdots \int_0^{l-w_1-\cdots-w_{ms-2}} dw_{m_s-1}} \\
&= \frac{\int_0^l dw_1 \int_0^{l-w_1} dw_2 \cdots \int_0^{l-w_1-\cdots-w_{ms-2}} \frac{1}{2l}(w_1^2+w_2^2+\cdots+w_{m_s-1}^2+w_{m_s}^2) dw_{m_s-1}}{\int_0^l dw_1 \int_0^{l-w_1} dw_2 \cdots \int_0^{l-w_1-\cdots-w_{ms-2}} dw_{m_s-1}} \\
&= \frac{1}{2l} \frac{\int_0^l dw_1 \int_0^{l-w_1} dw_2 \cdots \int_0^{l-w_1-\cdots-w_{ms-2}} [w_1^2+w_2^2+\cdots+w_{m_s-1}^2+(l-w_1-w_2-\cdots-w_{m_s-1})^2] dw_{m_s-1}}{\int_0^l dw_1 \int_0^{l-w_1} dw_2 \cdots \int_0^{l-w_1-\cdots-w_{ms-2}} dw_{m_s-1}} \\
&= \frac{1}{2l} \times \frac{Y_{m_s}}{V_{m_s}} \\
&= \frac{l}{m_s+1}, n \geq 2
\end{aligned}$$

where,

$$\begin{aligned}
Y_{m_s} &= \int_0^l dw_1 \int_0^{l-w_1} dw_2 \cdots \int_0^{l-w_1-\cdots-w_{ms-2}} [w_1^2+w_2^2+\cdots+w_{m_s-1}^2+(l-w_1-w_2-\cdots-w_{m_s-1})^2] dw_{m_s-1} \\
V_n &= \int_0^l dw_1 \int_0^{l-w_1} dw_2 \cdots \int_0^{l-w_1-\cdots-w_{ms-2}} dw_{m_s-1}
\end{aligned}$$

Eq. (4.2) is true if and if the following two equations are true,

$$Y_{m_s} = \frac{2}{m_s+1} \times \frac{l^{m_s+1}}{(m_s-1)!} \tag{A-1}$$

$$V_{m_s} = \frac{l^{m_s-1}}{(m_s-1)!} \tag{A-2}$$

Therefore, to prove Eq. (4.2), we only need to prove the two equations above. The following text will give the proof of Equation (A-1) and (A-2).

1) The proof of Y_{m_s}

We rewrite Y_{m_s} as n integrals,

$$\begin{aligned}
Y_n &= \int_0^l dw_1 \int_0^{l-w_1} dw_2 \cdots \int_0^{l-w_1-\cdots-w_{m_s-2}} w_1^2 dw_{m_s-1} \\
&+ \int_0^l dw_1 \int_0^{l-w_1} dw_2 \cdots \int_0^{l-w_1-\cdots-w_{m_s-2}} w_2^2 dw_{m_s-1} \\
&\cdots\cdots \\
&+ \int_0^l dw_1 \int_0^{l-w_1} dw_2 \cdots \int_0^{l-w_1-\cdots-w_{m_s-2}} w_{n-1}^2 dw_{n-1} \\
&+ \int_0^l dw_1 \int_0^{l-w_1} dw_2 \cdots \int_0^{l-w_1-\cdots-w_{m_s-2}} (l-w_1-w_2-\cdots-w_{m_s-1})^2 dw_{m_s-1}
\end{aligned}$$

First, we deduct the integrals of $w_i^2, i \in [1, m_s - 2]$ as follows.

$$
\begin{aligned}
& \int_0^l dw_1 \int_0^{l-w_1} dw_2 \cdots \int_0^{l-w_1-\cdots-w_{m_s-2}} w_i^2 dw_{m_s-1} \\
= & \frac{1}{(m_s-i-1)!} \int_0^l dw_1 \cdots \int_0^{l-w_1-\cdots-w_{i-1}} w_i^2 (l-w_1-\cdots-w_i)^{m_s-i-1} dw_i \\
= & \frac{1}{(m_s-i-1)!} \int_0^l dw_1 \cdots \int_0^{l-w_1-\cdots-w_{i-1}} \sum_{k=0}^{m_s-i-1} C_{m_s-i-1}^k (-1)^k w_i^{k+2} (l-w_1-\cdots-w_{i-1})^{m_s-i-k-1} dw_i \\
= & \frac{1}{(m_s-i-1)!} \int_0^l dw_1 \cdots \int_0^{l-w_1-\cdots-w_{i-2}} \sum_{k=0}^{m_s-i-1} C_{m_s-i-1}^k \frac{(-1)^k}{k+3} (l-w_1-\cdots-w_{i-1})^{m_s-i+2} dw_{i-1} \\
= & \frac{1}{(m_s-i-1)!} \sum_{k=0}^{m_s-i-1} C_{m_s-i-1}^k \frac{(-1)^k}{k+3} \int_0^l dw_1 \cdots \int_0^{l-w_1-\cdots-w_{i-2}} (l-w_1-\cdots-w_{i-1})^{m_s-i+2} dw_{i-1} \\
= & \frac{1}{(m_s-i-1)!} \sum_{k=0}^{m_s-i-1} C_{m_s-i-1}^k \frac{(-1)^k}{k+3} \times \frac{l^{m_s+1}}{(m_s-i+3) \times \cdots \times m_s \times (m_s+1)} \\
= & \frac{l^{m_s+1}}{(m_s+1)!} \sum_{k=0}^{m_s-i-1} C_{m_s-i-1}^k \frac{(-1)^k}{k+3} (m_s-i)(m_s-i+1)(m_s-i+2) \\
= & \frac{l^{m_s+1}}{(m_s+1)!} \sum_{k=0}^{m_s-i-1} (-1)^k C_{m_s-i+2}^{k+3} (k+1)(k+2) \\
= & \frac{l^{m_s+1}}{(m_s+1)!} S_{m_s}
\end{aligned}
$$

where,

$$
S_{m_s} = \sum_{k=0}^{m_s-i-1} (-1)^k C_{m_s-i+2}^{k+3} (k+1)(k+2)
$$

Next, we prove that $S_{m_s} = 2$ for $m_s \geq 2$. When $m_s = 2$, it is easy to see that

$S_2 = 2$. Assume $S_{m_s} = 2,$ for $m_s \geq 2$, Then

$$\begin{aligned}
S_{m_s+1} &= \sum_{k=0}^{m_s-i} (-1)^k C_{m_s-i+3}^{k+3}(k+1)(k+2) \\
&= (-1)^{m_s-i}(m_s-i+1)(m_s-i+2) + \sum_{k=0}^{m_s-i-1} (-1)^k (C_{m_s-i+2}^{k+3} + C_{m_s-i+2}^{k+2})(k+1)(k+2) \\
&= \sum_{k=0}^{m_s-i-1} (-1)^k C_{m_s-i+2}^{k+3}(k+1)(k+2) + \sum_{k=0}^{m_s-i} (-1)^k C_{m_s-i}^{k}(m_s-i+1)(m_s-i+2) \\
&= \sum_{k=0}^{m_s-i-1} (-1)^k C_{m_s-i+2}^{k+3}(k+1)(k+2) + 0 = S_{m_s} \\
&= 2
\end{aligned}$$

Therefore,

$$\int_0^l dw_1 \int_0^{l-w_1} dw_2 \cdots \int_0^{l-w_1-\cdots-w_{m_s-2}} w_i^2 dw_{m_s-1} = \frac{2l^{m_s+1}}{(m_s+1)!}, i \in [1, m_s-2]$$

Finally, we compute the integrals of $w_{m_s-1}^2$ and $(l - w_1 - w_2 - \cdots - w_{m_s-1})^2$ as follows.

$$\begin{aligned}
&\int_0^l dw_1 \int_0^{l-w_1} dw_2 \cdots \int_0^{l-w_1-\cdots-w_{m_s-2}} w_{m_s-1}^2 dw_{m_s-1} \\
&= \frac{2}{3!} \int_0^l dw_1 \int_0^{l-w_1} dw_2 \cdots \int_0^{l-w_1-\cdots-w_{m_s-3}} (l - w_1 - \cdots - w_{m_s-2})^3 dw_{m_s-2} \\
&= \frac{2}{4!} \int_0^l dw_1 \int_0^{l-w_1} dw_2 \cdots \int_0^{l-w_1-\cdots-w_{m_s-4}} (l - w_1 - \cdots - w_{m_s-3})^4 dw_{m_s-3} \\
&\cdots\cdots \\
&= \frac{2}{m_s!} \int_0^l (l - w_1)^{m_s} dw_1 \\
&= \frac{2l^{m_s+1}}{(m_s+1)!}
\end{aligned}$$

Similarly, it can be proved,

$$\int_0^l dw_1 \int_0^{l-w_1} dw_2 \cdots \int_0^{l-w_1-\cdots-w_{m_s-2}} (l-w_1-w_2-\cdots-w_{m_s-1})^2 dw_{m_s-1} = \frac{2l^{m_s+1}}{(m_s+1)!}$$

In conclusion, in Equation (A-1), the integral of each integrand, i.e., $w_1^2, w_2^2, \ldots, w_{m_s-1}^2, (l-w_1-w_2-\cdots-w_{m_s-1})^2$, is equal to $\frac{2l^{m_s+1}}{(m_s+1)!}$. Therefore,

$$Y_{m_s} = m_s \times \frac{2l^{m_s+1}}{(m_s+1)!} = \frac{2}{m_s+1} \times \frac{l^{m_s+1}}{(m_s-1)!}$$

2) The proof of V_{m_s}

$$\begin{aligned}
V_{m_s} &= \int_0^l dw_1 \int_0^{l-w_1} dw_2 \cdots \int_0^{l-w_1-\cdots-w_{m_s-2}} dw_{m_s-1} \\
&= \frac{1}{1!} \int_0^l dw_1 \int_0^{l-w_1} dw_2 \cdots \int_0^{l-w_1-\cdots-w_{m_s-3}} (l-w_1-\cdots-w_{m_s-2}) dw_{m_s-2} \\
&= \frac{1}{2!} \int_0^l dw_1 \int_0^{l-w_1} dw_2 \cdots \int_0^{l-w_1-\cdots-w_{m_s-4}} (l-w_1-\cdots-w_{m_s-3})^2 dw_{m_s-3} \\
&= \frac{1}{3!} \int_0^l dw_1 \int_0^{l-w_1} dw_2 \cdots \int_0^{l-w_1-\cdots-w_{m_s-5}} (l-w_1-\cdots-w_{m_s-4})^3 dw_{m_s-4}
\end{aligned}$$

$$\cdots\cdots$$

$$\begin{aligned}
&= \frac{1}{(m_s-2)!} \int_0^l (l-w_1)^{m_s-2} dw_1 \\
&= \frac{l^{m_s-1}}{(m_s-1)!}
\end{aligned}$$

Bibliography

[1] 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.

[2] 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.

[3] T. Small and Z. J. Haas, "The shared wireless infostation model - a new ad hoc networking paradigm (or where there is a whale, there is a way)," in *ACM MOBIHOC*, (Annapolis, Maryland), June 2003.

[4] T. Small, Z. Haas, A. Purgue, and K. Fristrup, "A sensor network for biological data acquisition," in *CRC*, 2004.

[5] P. Karras and N. Mamoulis, "Detecting the direction of motion in a binary sensor network," in *Proceedings of the IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing (SUTC)*, June 2006.

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

[7] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar, "Multilevel hypergraph partitioning: Applications in VLSI domain," tech. rep., 1997.

[8] G. Karypis and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs," *SIAM Journal on Scientific Computing*, vol. 20, no. 1, pp. 359–392, 1998.

[9] J. Edmonds and E. Johnson, "Euler tours and the chinese postman problem," *Mathematical Programming*, vol. 5, pp. 88–124, 1973.

[10] D. Ahr and G. Reinelt, "A tabu search algorithm for the min-max k-chinese postman problem," *Comput. Oper. Res.*, vol. 33, no. 12, pp. 3403–3422, 2006.

[11] G. Groves and J. van Vuuren, "Efficient heuristics for the rural postman problem," *ORiON*, vol. 21, no. 1, pp. 33–51, 2005.

[12] 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*, (Piscataway, NJ, USA), pp. 175–181, IEEE Press, 1982.

[13] B. Hendrickson and R. Leland, "An improved spectral graph partitioning algorithm for mapping parallel computations," *SIAM Journal on Scientific Computing*, vol. 16, no. 2, pp. 452–469, 1995.

[14] A. Pothen, H. D. Simon, and K.-P. Liou, "Partitioning sparse matrices with eigenvectors of graphs," *SIAM J. Matrix Anal. Appl.*, vol. 11, no. 3, pp. 430–452, 1990.

[15] A. Pothen, H. D. Simon, L. Wang, and S. T. Barnard, "Towards a fast implementation of spectral nested dissection," in *Supercomputing '92: Proceedings of the 1992 ACM/IEEE conference on Supercomputing*, (Los Alamitos, CA, USA), pp. 42–51, IEEE Computer Society Press, 1992.

[16] 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*, (Los Alamitos, CA, USA), pp. 538–547, IEEE Computer Society Press, 1991.

[17] M. T. Heath and P. Raghavan, "A Cartesian parallel nested dissection algorithm," *SIAM Journal on Matrix Analysis and Applications*, vol. 16, no. 1, pp. 235–253, 1995.

[18] G. L. Miller, S.-H. Teng, W. Thurston, and S. A. Vavasis, "Automatic mesh partitioning in sparse matrix computations: Graph theory issues and algorithms." A. George, J. R. Gilbert, and J. W.-H Liu, eds (an IMA workshop volume), Springer-Verlag, New York, 1993.

[19] T. Bui and C. Jones, "A heuristic for reducing fill in sparse matrix factorization," in *Proc. Sixth SIAM Conf. Parallel Processing for Scientific Computing*, pp. 445–452, 1993.

[20] C.-K. Cheng and Y.-C. A. Wei, “An improved two-way partitioning algorithm with stable performance,” in *IEEE Transactions on Computer Aided Design*, vol. 10, pp. 1502–1511, December 1991.

[21] J. Garbers, H. Promel, and A. Steger, “Finding clusters in vlsi circuits,” in *Proc. Intl Conf. Computer-Aided Design*, pp. 520–523, Nov 1990.

[22] B. Hendrickson and R. W. Leland, “A multi-level algorithm for partitioning graphs,” in *Supercomputing*, 1995.

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

[24] S. Even and R. E. Tarjan, “Network flow and testing graph connectivity,” *SIAM J. Computing*, pp. 507–518, 1975.

[25] Z. Galil and G. F. Italiano, “Reducing edge connectivity to vertex connectivity,” *ACM SIGACT News 22*, pp. 57–61, 1991.

[26] M. R. Henzinger, S. Rao, and H. N. Gabow, “Computing vertex connectivity: new bounds from old techniques,” in *Proc. 37th IEEE F. O. C. S.*, pp. 462–471, 1996.

[27] D. W. Matula, “Determining edge connectivity in o(mn),” in *Proceedings, 28th Symp. on Foundations of Computer Science*, pp. 249–251, 1987.

[28] C. S. Orloff, “A fundamental problem in vehicle routing,” *Networks*, vol. 4, pp. 35–64, 1974.

[29] G. G and L. G, “A branchvandvcut algorithm for the undirected rural postman problem,” *Mathematical Programming*, vol. 87, pp. 467–481, 2000.

[30] H. A, L. G, and N.-H. P, “Improvement procedures for the undirected rural postman problem,” *INFORMS Journal on Computing*, vol. 11, no. 1, pp. 53–62, 1999.

[31] F. de C’ordoba P, G. R. LM, and S. JM, “A heuristic algorithm based on monte carlo methods for the rural postman problem,” *Computers and Operations Research*, vol. 25, no. 12, pp. 1097–1106, 1998.

[32] L. AN, *Polyhedral results for some constrained arc routing problems*. PhD thesis, Lancaster University, Lancaster, 1996.

[33] G. N. Frederickson, M. S. Hecht, and C. E. Kim, “Approximation algorithms for some routing problems,” *SIAM Journal of Computing*, vol. 7, pp. 178–193, May 1978.

[34] J. Degenhardt, “An ant-algorithm for the balanced k-chinese postmen problem,” in *Operations Research Conference 2004 (OR 04)*, Tilburg University, September 2004.

[35] R. S. Garfinkel and I. R. Webb, “On crossings, the crossing postman problem, and the rural postman problem,” *Networks*, vol. 34, pp. 173–180, 1999.

[36] J. Araoz, E. Fernandez, and C. Zoltan, “Privatized rural postman problems,” *Comput. Oper. Res.*, vol. 33, no. 12, pp. 3432–3449, 2006.

[37] D. Feillet, P. Dejax, and M. Gendreau, “Traveling salesman problems with profits,” *Transportation Science*, vol. 39, no. 2, pp. 188–205, 2005.