UNIVERSITY OF CALIFORNIA, SAN DIEGO

Controlled Mobility in Sensor Networks

A dissertation submitted in partial satisfaction of the
requirements for the degree
Doctor of Philosophy

in

Computer Science

by

Ryo Sugihara

2009

The dissertation of Ryo Sugihara is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

Chair

University of California, San Diego

2009

DEDICATION

To my wife and two daughters

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

First of all, I would like to thank my advisor, Professor Rajesh Gupta, for his guidance, support, encouragements, and patience. His deep insight brought me a lot of ideas that were invaluable for all of my work.

I am also grateful for Professors Sanjoy Dasgupta, Massimo Franceschetti, William Hodgkiss, and George Varghese for serving on my doctoral committee and giving me valuable feedbacks. Especially for Professor Varghese, I was fortunate to have an opportunity to work as the TA for his class, from which I have learned many things about teaching.

I also would like to thank my former advisor Professor Andrew Chien, who recommended me to pursue the research on sensor networks.

I have learned a lot from my fellow students at Microelectronics Embedded Systems Lab. My appreciation goes to Yuvraj Agarwal, Joel Coburn, Arup De, Frederic Doucet, Zhong-Yi Jin, Sudipta Kundu, Kaisen Lin, and Thomas Weng. I also thank all the people at the old Concurrent Systems Architecture Group, especially to Jerry Chou, Dionysious Logothesis, and Han Suk Kim, all of whom have been great friends of mine.

I would like to thank IBM Tokyo Research Laboratory for the financial support that made my graduate study at UCSD possible. I also thank the former director, Dr. Hiroshi Maruyama, for his understanding and continuous encouragements.

I would like to thank Professor Minming Li at City University of Hong Kong for various suggestions on my early work during his visit to UCSD in 2008.

Last, but not least, I would like to thank my family. My wife, Chiaki, and my daughters, Hana and Aya, have been a great comfort for me when I was stressed out. I also thank my parents in Japan who have supported me all the time.

Papers included in this dissertation

Chapter 4 and 5, in part, have been submitted for publication as “Complexity of Motion Planning of Data Mule for Data Collection in Wireless Sensor Networks” by Ryo Sugihara and Rajesh K. Gupta in Theoretical Computer Science [SG09a]. The dissertation author was the primary investigator and author of this paper.

Chapter 4, 8, and 9, in part, have been submitted for publication as “Speed Control and Scheduling of Data Mules in Sensor Networks” by Ryo Sugihara and Rajesh K. Gupta in ACM Transactions on Sensor Networks [SG09e]. The dissertation author was the primary investigator and author of this paper.

Chapter 5, in part, has been accepted for publication as “Optimal Speed Control of Mobile Node for Data Collection in Sensor Networks” by Ryo Sugihara and Rajesh K. Gupta in IEEE Transactions on Mobile Computing [SG09b]. The dissertation author was the primary investigator and author of this paper.

Chapter 6, in part, has been published as “Improving the Data Delivery Latency in Sensor Networks with Controlled Mobility” by Ryo Sugihara and Rajesh K. Gupta in the proceedings of the 4th IEEE International Conference on Distributed Computing in Sensor Systems (DCOSS) [SG08a]. The dissertation author was the primary investigator and author of this paper.

Chapter 6, 8, and 9, in part, have been submitted for publication as “Path Planning of Data Mules in Sensor Networks” by Ryo Sugihara and Rajesh K. Gupta in ACM Transactions on Sensor Networks [SG09d]. The dissertation author was the primary investigator and author of this paper.

Chapter 7, in part, has been published as “Optimizing Energy-Latency Trade-off in Sensor Networks with Controlled Mobility” by Ryo Sugihara and Rajesh K. Gupta in the proceedings of the 28th Annual Conference of the IEEE Communications Society (INFOCOM) Mini-conference [SG09c]. The dissertation author was the primary investigator and author of this paper.

VITA

1997	Bachelor of Engineering, University of Tokyo, Japan
1999	Master of Engineering, University of Tokyo, Japan
2009	Doctor of Philosophy, University of California, San Diego, USA

PUBLICATIONS

Ryo Sugihara and Rajesh K. Gupta, "Path Planning of Data Mules in Sensor Networks" under submission to *ACM Transactions on Sensor Networks*

Ryo Sugihara and Rajesh K. Gupta, "Complexity of Motion Planning of Data Mule for Data Collection in Wireless Sensor Networks" under submission to *Theoretical Computer Science*

Ryo Sugihara and Rajesh K. Gupta, "Speed Control and Scheduling of Data Mules in Sensor Networks" under submission to *ACM Transactions on Sensor Networks*

Ryo Sugihara and Rajesh K. Gupta, "Optimal Speed Control of Mobile Node for Data Collection in Sensor Networks" accepted for publication in *IEEE Transactions on Mobile Computing*

Ryo Sugihara and Rajesh K. Gupta, "Optimizing Energy-Latency Trade-off in Sensor Networks with Controlled Mobility" in *Proceedings of the 28th Annual Conference of the IEEE Communications Society (INFOCOM) Mini-conference*, Apr. 2009.

Ryo Sugihara and Rajesh K. Gupta, "Improving the Data Delivery Latency in Sensor Networks with Controlled Mobility" in *Proceedings of the 4th IEEE International Conference on Distributed Computing in Sensor Systems (DCOSS)*, Jun. 2008. **Best Paper Award in Systems Track**

Ryo Sugihara and Rajesh K. Gupta, "Programming Models for Sensor Networks: A Survey" *ACM Transactions on Sensor Networks*, vol.4, issue 2, Mar. 2008.

ABSTRACT OF THE DISSERTATION

Controlled Mobility in Sensor Networks

by

Ryo Sugihara

Doctor of Philosophy in Computer Science

University of California, San Diego, 2009

Professor Rajesh K. Gupta, Chair

We consider the problem of collecting data from stationary sensor nodes using controllable mobile nodes ("data mules") via wireless communication. Whereas the use of data mules can significantly reduce the energy consumption at sensor nodes, a drawback is an increased data delivery latency. Reducing the latency through optimizing the motion of data mules is critical for this approach to be useful.

Since optimizing the motion of data mules is a hard problem in general, previous studies simplified the problem by using basic models for mobility and communications. These simplifications often lead to suboptimal solutions and also to algorithms that are only applicable to limited scenarios.

To address these limitations, in this dissertation we present a problem framework that we call the Data Mule Scheduling (DMS) problem. The DMS problem captures the motion planning problem as the one composed of loosely connected subproblems.

We first study the one dimensional case of the DMS (1-D DMS) problem, representing the speed control of data mule and the scheduling of data collection jobs. For different mobility models, we present optimal algorithms or non-existence of them, prove NP-hardness, and design an approximation algorithm to determine speed control and job schedule. We also design a heuristic algorithm for hard cases and demonstrate the good performance through simulation experiments by comparing with theoretical lower bounds. Furthermore, we show how the mathematical formulation of the 1-D DMS problem can be used as a proxy to solve speed scaling problems for dynamic power management of processors.

For the two dimensional case that includes the planning of data mule's path, we formulate the path selection problem as a graph problem and design an approximation algorithm. We also explore the combination of data mule and multihop forwarding and demonstrate how this hybrid approach enable a flexible trade-off between node energy consumption and data delivery latency.

In the end, we present some extensions of the DMS problem framework for the cases of multiple data mules and partially-known communication range. These cases allow the DMS framework to be applied to larger range of application scenarios under realistic radio environments.

Chapter 1

Introduction

Sensor networks refer to a network of small devices that are equipped with sensors and with communication and processing capabilities. Technology advances have made it possible to make sensor nodes small enough to be embedded in the environments for *in situ* measurements. Sensor networks are rapidly growing for their large potential in various application areas including environmental monitoring, target tracking, structural health monitoring, and so on.

Majority of sensor network applications are *data-collection* applications, in which the objective is to collect the data from sensor nodes to a base station. The base station serves as a gateway for the sensor nodes and may represent a central server where the end application resides or an embedded processing node in a hierarchically organized network.

1.1 Emerging Sensor Network Applications

We first introduce actual application examples and show the data-collection applications are a majority of sensor network applications. Then we describe that multihop forwarding is a commonly-used approach for data collection in these applications, and further examine important issues in using multihop forwarding for data collection.

1.1.1 Application Examples

Habitat monitoring is one of the earliest applications of sensor networks. In Great Duck Island [SOP$^+$04], researchers monitored the behavior of petrels, especially about how they use burrows both in short-term and long-term periods. They also monitored the environmental parameters inside and outside of burrows. Another example is the ZebraNet project [JOW$^+$02], in which the behavior of zebras including long-range migration, inter-species interactions, and nocturnal behavior is monitored using tracking collars. Such monitoring applications yield valuable data for the field researchers.

Environmental monitoring is another area that sensor networks have been widely used. Examples include meteorological and hydrologic processes at high altitudes [LCD03], long-term glacial movement [MHO04], temperature and humidity in the forest [BRY$^+$04, TPS$^+$05], and volcanic activities [WALR$^+$06]. Structural monitoring [XRC$^+$04] is to collect and to analyze structural response to ambient or forced excitation by using accelerometer.

A characteristic application is target tracking, in which the objective is to localize a target by trilateration and other techniques using multiple sensors capable of measuring distance or bearing angle of the target (e.g., [CHZ02, PPK03, HKS$^+$04, SML$^+$04]). As we have pointed out in our earlier survey [SG08b], target tracking applications are qualitatively different from habitat/environmental monitoring applications in the sense that it intrinsically requires collaborative information processing among sensors.

In all of these applications, we can observe that the high-level objective is to bring the sensor data from a distributed field of sensors to a base station. In habitat/environmental monitoring, it is for domain-specific researchers to further analyze the data, and in target tracking, it is for the users to take appropriate actions using the information on the detected target. For this reason, we call them *data-collection applications*.

1.1.2 Approaches for Data Collection and Their Issues

Traditionally, data collection has been done using multihop forwarding among the sensor nodes. This is because the wireless communication range of each node is not long enough to send the data directly to the base station. Another reason is that, even if it is possible for a node to send the data directly to the base station, it is often more energy efficient to use multihop communications instead, which is derived by the convex relation between the wireless range and the energy consumption (e.g., [HY05]). The energy issue is very important for sensor networks, since they are usually deployed in remote areas where wire power is hard to obtain. As a result, sensor nodes are usually driven by batteries and they need to avoid energy dissipation as much as possible to realize long-term monitoring of phenomena of interest.

Although multihop forwarding is more energy-efficient than long range communications, there are some issues. The first is that multihop forwarding cannot be used when the network is disconnected, i.e., there is no multihop route to the base station. In addition for sparse networks, even if the network is not disconnected, energy consumption is high because the distance of each hop tends to be long. A possible workaround for these cases is to deploy additional nodes to maintain the connectivity and reduce each hop distance, but this may not always be possible.

The second issue is that energy consumption tends to be unbalanced among the nodes. This is caused by the fact that the nodes near the base station need to forward a large amount of data received from the ones farther than them. Even if we use duty-cycling, the nodes near the base station need to be more active than others. Since the communication is one of the most power-consuming operation for a node, these nodes tend to run out of battery sooner than other nodes. Then, even if most of the nodes are still active, the application is inoperable because the network is disconnected. Some techniques try to balance the energy consumption by randomly rotating the clusterhead [HCB00, LR02, BC03, YF04], but the overall tendency of unbalance does not change.

Finally, the nodes need to be capable enough to be able to handle multihop communication pattern. This becomes a problem for some applications such as [TMF$^+$07], which uses RFID-based sensor nodes.

1.2 Use of Controlled Mobility in Sensor Networks

An alternative and relatively new approach for data collection from sensor networks is to exploit mobility. There are several types of mobility, but our focus in this dissertation is on *controlled mobility*, where the users can determine the motion of mobile nodes. Among possible usage of controlled mobility, we discuss *data mule approaches* for data collection.

1.2.1 Data Mule Approaches for Data Collection

Suppose, in the network, there are one or more mobile agents that are capable of communicating with sensor nodes and carrying data. The basic idea of data mule approaches is to use these mobile agents as routers from sensor nodes to the base station.

More specifically, a mobile node ("data mule") starts from the base station, travels across the sensor field while collecting data from sensor nodes, and comes back to the base station to deposit the data.

Note that we focus on the case that the data mules are controllable. Accordingly we use the term *data mule* solely to refer to controllable mobile nodes for data collection. Just as a reference, similar approach is possible without controlled mobility: in fact the term "data mules" was coined by Shah et al. [SRJB03] to refer to randomly mobile nodes used for data collection.

Data mule approaches have several benefits that overcome the issues in the traditional sensor networks. For example, in data mule approaches each sensor node can conserve a significant amount of energy, since a node can send the data to the data mule when it is traveling nearby and also there is no need to forward other nodes' data. Disconnected networks or sparse networks are not an issue for data mule approaches, since the communications among nodes are not necessary. Furthermore, by eliminating the need for multihop forwarding, data mule approaches can be applicable to the application scenarios where the sensor nodes do not have enough computation resources.

Some recent sensor network applications use such data mules for data collection, e.g., a robot in underwater environmental monitoring [VKR+05] and a UAV (unmanned aerial vehicle) in structural health monitoring [TMF+07].

1.2.2 Need for Optimizing the Motion

A drawback in data mule approaches is an increased data delivery latency, i.e., the latency from the data is generated at a sensor until it is delivered to the base station. This is because the latency is mostly governed by the physical motion of data mule, which is relatively slow compared to the speed of data transmitted in multihop forwarding.

For this reason, optimizing the motion of data mules is important. For the applications that multihop forwarding is also an option, data mule approaches give another design choice that has a different characteristic in energy-latency trade-off. Improving the latency through the motion optimization leads to a better energy-latency trade-off. Optimizing the data mules' motion is even more important for the applications in which multihop forwarding approach is not possible, since data mule approaches are the only choice for this case.

1.2.3 Issues in Previous Approaches

Optimizing the motion of data mule is a hard problem in general, since we need to find a trajectory satisfying both spatial constraints (i.e., wireless communication range) and temporal constraints (i.e., time required for data collection). The hardness is indicated by the simple case where the communication range is zero and data collection time at each node is negligible is equivalent to the Traveling Salesman Problem (TSP), which is an NP-hard problem.

To deal with the hardness, previous studies on data mule approaches [KSJ$^+$04, SRS04, MY06, XWXJ07] simplified the problem by using simple models for mobility and communications. Some examples include zero communication range (e.g., [SRS04, SRS07]) and constant speed data mules (e.g., [MY06, MY07, XWXJ07]). These simplifications result in loss of opportunity for further optimization when the data mule can communicate with nodes remotely or can change the speed.

Furthermore, the algorithms presented in these studies are often tailored for specific settings: e.g., certain node deployments [MY06, MY07] or existence of forwarding nodes [XWXJ07]. This makes it hard to use these algorithms for similar but different application scenarios.

1.3 Our Contributions

This dissertation investigates the use of controlled mobility in sensor networks, specifically on how to optimize the motion of data mules to improve the data delivery latency. While the overall nature of our contribution is mostly formal, focused on the combinatoric nature of the problem, we provide a framework that allows us to systematically remove simplifying assumptions as needed and to derive solutions that apply to practical scenarios for using data mules.

Our first contribution is the problem framework for optimizing the motion of data mules. We have designed the problem framework so that it is comprehensive enough to express many different problem settings and allows problem formulations in a tractable way. We call this problem framework the *Data Mule Scheduling* (DMS) problem. A key idea of the DMS problem is to divide the original problem into a set of loosely connected subproblems so that they can be solved independently. Another key idea is to cast the problem as a generalized scheduling problem. The scheduling problem is unique in the sense that each job has location constraints as well as time constraints. We also show how the DMS framework can be extended for several different cases including hybrid case with multihop forwarding, multiple data mules, and partially-known communication ranges.

The second contribution is building a theoretical foundation of the DMS problem. For one dimensional case of the DMS problem, we have done a thorough analysis on computational complexity of the problem for different motion constraints. The analysis includes designing several optimal algorithms, proving NP-hardness for important cases, and designing an approximation algorithm for these cases. Further we show an interesting connection with speed scaling problems for dynamic power management of processors and also present how an algorithm for the DMS problem can be applied to speed scaling problems. For the two dimensional case, we formulate the path selection problem as a graph problem called *Label-Covering Tour*, prove the NP-hardness, and design an approximation algorithm.

Our third contribution is on performance evaluations through realistic scenarios. We have implemented our algorithms on MATLAB and ns2 network simulator [ns2]. We also analyze the lower bounds of the problems through the relaxations to linear programming (LP) and semidefinite programming (SDP) and compare the simulation results with these lower bounds to evaluate the performance.

1.4 Organization of the Dissertation

Figure 1.1 shows the organization of this dissertation. Chapter 2 discusses the background of the research and introduces related work. In Chapter 3, we introduce the Data Mule Scheduling (DMS) problem as a problem framework for optimally controlling a data mule. We discuss one dimensional case of the problem (1-D DMS) first in Chapter 4 and 5. Specifically, the basic cases with simple mobility models are discussed in Chapter 4 and more general case with acceleration constraint is discussed in Chapter 5. The path selection subproblem is discussed in Chapter 6. In Chapter 7, the hybrid approach of data mule and multihop forwarding is discussed In Chapter 8 and 9, we explore several extensions of the DMS framework. These extensions include the case of multiple data mules and the case when the communication range is known only partially. Finally, Chapter 10 concludes the dissertation.

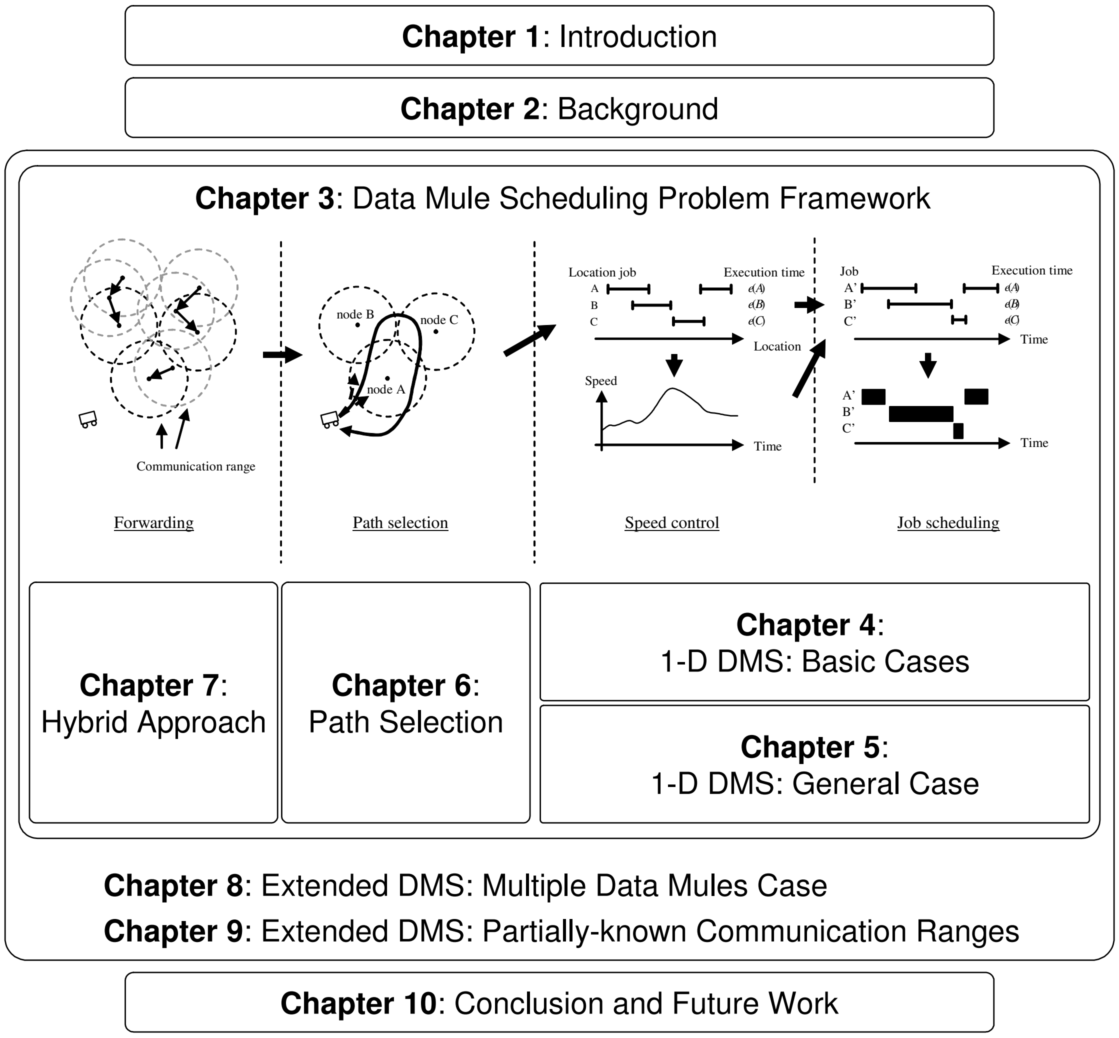

Chapter 1: Introduction
Chapter 2: Background
Chapter 3: Data Mule Scheduling Problem Framework
Communication range
node B
node C
node A
Location job
Execution time
A
B
C
e(A)
e(B)
e(C)
Location
Speed
Time
Job
A'
B'
C'
Forwarding
Path selection
Speed control
Job scheduling
Chapter 7: Hybrid Approach
Chapter 6: Path Selection
Chapter 4: 1-D DMS: Basic Cases
Chapter 5: 1-D DMS: General Case
Chapter 8: Extended DMS: Multiple Data Mules Case
Chapter 9: Extended DMS: Partially-known Communication Ranges
Chapter 10: Conclusion and Future Work

Figure 1.1: Organization of the dissertation

Chapter 2

Problem Background

In this chapter we first look into the mobility in sensor networks, particularly about how different types of mobility are handled and also exploited in previous studies. Then we present related work on data mule approaches. We introduce a structural health monitoring application as an application example that uses a data mule for data collection. Finally we analyze previously proposed systems and algorithms for data mule approaches, specifically about their application models and assumptions.

2.1 Mobility in Sensor Networks

We first classify the mobility and discuss the effect on wireless networking. Consider a sensor network in which a source node S needs to send data to a target destination node D, as shown in Figure 2.1. Although the communication may be direct from S to D, there are usually one or more intermediary nodes I. Based on the classification in [KSJ$^+$04], we can treat node mobility as one of the four basic types:

Stationary A node does not move from its original location.

Random A node moves in a random fashion. The motion may follow a probabilistic model characterized by some parameters.

Predictable A node's future position is known but cannot be changed.

Controlled A node can be controlled by a user. There may be constraints on the motion such as maximum speed and maximum acceleration.

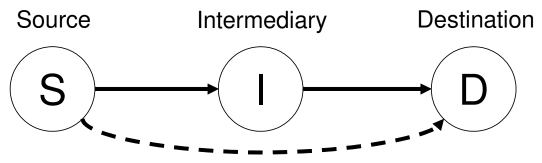


Figure 2.1: Basic communication pattern in wireless networks

Table 2.1: Communication in wireless networks classified by the type of mobility

	Source	Intermediary	Destination	Approach
0	stationary	stationary	stationary	Multihop forwarding
1	random	random	random	Epidemic Routing [VB00]
2	random	random	controlled	ZebraNet [JOW$^+$02]
3	random	random	stationary	SWIM [SH03]
4	stationary	random	stationary	DataMULEs [SRJB03], SENMA [TZA03]
5	stationary	predictable	stationary	Chakrabarti et al. [CSA03], BriMon [CRM$^+$08]
6	stationary	stationary	controlled	Mobile base station [GDPV03, LH05]
7	stationary	controlled	stationary	Pure data mule approach [ZA03, SRS04, ZAZ05, GBBE06]
				In this dissertation: Chapter 4-6
8	stationary	stationary + controlled	stationary	Hybrid data mule approach [KSJ$^+$04, MY06, MY07, JSS05, XWXJ07, XWJL08]
				In this dissertation: Chapter 7

Referring to Figure 2.1, in a sensor network application, any of the source, destination, and intermediary nodes can be mobile, leading to various different possibilities. Of these, the most interesting are listed in Table 2.1. We have organized this table to indicate how previous work in this area relates to the work presented in this dissertation. Let us now examine the previous related work in depth.

2.1.1 Random Mobility

Random mobility has been extensively studied in the context of mobile ad hoc networks (MANETs) rather than sensor networks. A general problem is to how to deliver the data from the source to the destination when we need to make use of intermediary nodes that are randomly moving (Table 2.1, Case 1). A natural approach is flooding, in which a node that received the data tries to forward that to whoever it encounters, hoping that the data is eventually delivered to the destination. This idea is used in Epidemic Routing [VB00]. In this case, mobility is mostly a nuisance that hampers efficient data transmission.

On the other hand, even random mobility can be exploited for improving the network performance. Theoretically, Grossglauser and Tse [GT02] showed the network capacity is increased by having randomly moving mobile nodes compared to the static network case discussed by Gupta and Kumar [GK00].

In sensor network scenarios, there are some studies on collecting data from randomly mobile sources (Table 2.1, Case 2). ZebraNet [JOW$^+$02] builds upon a similar idea as Epidemic Routing to improve the rate of successfully collected data in a habitat monitoring application. Specifically, a sensor node is attached to a zebra and collected data is copied to another zebra when they encounter, so that the researchers can collect the data for many zebras when they encounter a zebra. The Shared Wireless Infostation Model (SWIM) [SH03] is an architecture with a similar objective as ZebraNet but uses stationary nodes for the destination (Table 2.1, Case 3). In their scenario, sensor nodes move randomly while copying their data among them and the stationary node gets the data when a sensor node approaches it.

Closer to our work is the case with stationary sources and randomly mobile intermediaries (Table 2.1, Case 4). Shah et al. [SRJB03] proposed a network architecture for data collection using random mobility. They proposed a three-tier architecture having mobile entities called Data MULEs (Mobile Ubiquitous LAN Extensions) in the middle tier on top of stationary sensors under wired access points. Similarly as the data mule approaches with controlled mobility, Data MULEs collect data from sensor nodes when they are in close proximity and deposit it at the wired access points. A difference is that their scheme is rather optimistic, since they assume random mobility. SENMA [TZA03] is a similar architecture but the focus is more on PHY and MAC layer designs.

2.1.2 Predictable Mobility

Predictable mobility refers to the case when the motion is known but cannot be changed. A representative case is the intermediary with predictable mobility, which applies to the case in which public transportation such as train or bus is used for a vehicle for data.

Chakrabarti et al. [CSA03] studied the case of an intermediary with predictable mobility (Table 2.1, Case 5). They analyzed the gain in power consumption of sensor nodes by modeling the data collection process as a queueing system. BriMon [CRM$^+$08] is an application example that uses predictable mobility. The objective is BriMon is to monitor the structural health of the train bridges. Sensor nodes are attached to a bridge to measure vibrations after a train passes on the bridge, and the data is collected by the next train.

2.1.3 Controlled Mobility

Ekici et al. [EGB06] classified the approaches of exploiting controlled mobility into the following three types:

- MBS (Mobile base station)-based solutions
- MDC (Mobile data collector)-based solutions
- Rendezvous-based solutions

where the latter two are called data mule approaches in our terminology.

In MBS-based solutions, there is a mobile sink that can change the position (Table 2.1, Case 6). The data is delivered to the mobile sink using multihop forwarding among stationary nodes. The problem is mostly about finding the optimal motion of the mobile sink to balance the energy consumption.

MDC-based solutions are equal to data mule approaches: using a single hop communication to a data mule (Table 2.1, Case 7). Rendezvous-based solutions are a hybrid of multihop forwarding and data mule approaches (Table 2.1, Case 8). In this case the intermediary is stationary for multihop forwarding among nodes and controlled as for data mule. We also treat this case as data mule approaches, but to distinguish it from MDC-based solutions, we call MDC-based solutions “pure” data mule approaches, whereas Rendezvous-based solutions are called “hybrid” data mule approaches. Previous work in these approaches are discussed in detail in the next section.

2.2 Related Work on Data Mule Approaches

2.2.1 Application Example: Structural Health Monitoring

Our problem formulation is based on our experience with the example application described in [TMF$^+$07]. It is a structural health monitoring (SHM) application to do post-event (e.g., earthquakes) assessments for large-scale civil infrastructure such as bridges. Automated damage assessment using sensor systems is much more efficient and reliable than human visual inspections.

In this application, the sensor nodes operate completely passively and do not carry batteries, for the sake of long-term measurement and higher maintainability. Upon data collection, an external mobile intermediary provides energy to each node via microwave transmission, wakes it up, and collects data from it. The prototype system uses a radio-controlled helicopter as the mobile intermediary that is either remotely-piloted or GPS-programmed. Each sensor node is equipped with ATmega128L microcontroller, a 2.4GHz XBee radio, antennas for transmission/reception, and a supercapacitor to store the energy. Each node has two types of sensors. One is a piezoelectric sensing element integrated with nuts and washers to check if the bolt has loosened. The other is capacitive-based sensors for measuring peak displacement and bolt preload. Since the size of data from these sensors are small, communication time is almost negligible; however, it takes a few minutes to charge a supercapacitor through microwave transmission. In their latest prototype, the charging time is reduced to 95 seconds in the lab experiments and 270 seconds in the field experiments [MFL$^+$08].

The data collected by the UAV is brought back to the base station and analyzed by researchers using statistical techniques for damage existence and its location/type. Since the primary purpose of this application is to assess the safety of large civil structures after a disaster such as an earthquake, every process including data collection and analysis needs to be as quick as possible for prompt recovery. Furthermore, shorter travel time is required in view of the limited fuel on the helicopter.

Thus the goal of our formulation is to achieve data collection from spatially distributed wireless sensors in the minimum amount of time. It also provides another reason for using controlled mobility instead of multihop forwarding approach: simply because the SHM sensors are not capable of doing multihop communication. Furthermore, use of UAVs implies the need for more precise mobility model that takes acceleration constraint into consideration, as opposed to the simple “move or stop“ model used in majority of the related work.

2.2.2 Systems and Algorithms for Data Mule Approaches

We introduce previously proposed systems and algorithms for data mule approaches. Following the classification in Table 2.1, we first discuss pure data mule approaches and then discuss hybrid data mule approaches.

Pure Data Mule Approaches

Zhao and Ammar [ZA03] study the use of controlled mobility in mobile ad hoc networks. They assume a controllable mobile node (called “ferry”) that mediates communications between sparsely deployed stationary nodes. They considered the extent of wireless communication range to optimize the movement, but the path selection is done based on TSP-like formulation and thus a ferry visits the exact locations of all nodes. The speed of ferry is basically constant but can be reduced when it is necessary to communicate more data with a node. They have formulated the problem of minimizing message delivery delay as a linear programming problem. They extended their work to multiple data mules case in [ZAZ05] and presented heuristic algorithms.

Somasundara et al. [SRS04] studied the problem of choosing the path of a data mule that traverses through a sensor field where sensors generate data at a given rate. In their formulation, the data mule moves at a constant speed. Their formulation also requires the data mule to visit the exact location of each sensor to collect data. They designed heuristic algorithms to find a path that minimizes the buffer overflow at each sensor node. Gu et al. [GBBE06] presented an improved algorithm for the same problem settings as [SRS04].

For multiple data mules case, Somasundara et al. [SRS07] studied the path selection problem in a similar setting as [SRS04]. Speed control is not discussed and they proposed a heuristic algorithm based on the formulation as the Vehicle Routing Problem (VRP).

Hybrid Data Mule Approaches

Kansal et al. [KSJ$^+$04] presented an algorithm for controlling the speed of a data mule. Assuming that the data mule periodically travels across the sensor field along a fixed path, they presented an adaptive algorithm. The objective was to maximize the amount of collected data under the constraint of maximum latency (i.e. travel time for one period). A notable thing is that they made no assumption about the communication range, since their focus was on designing a robust communication infrastructure. Instead, the data mule dynamically finds the nodes from which it can collect data. To improve the data collection performance, the data mule changes the behavior based on the past performance. Specifically, the data mule slows down when it encounters the nodes from which data collection has not been very successful in the previous period. They also designed a communication protocol based on directed diffusion [IGE00], in which the data mule issues interest messages to nodes and nodes try to forward the data toward the source of the interest message. In this way the data mule can collect data from the nodes that are not in the direct communication range. They evaluated their algorithm on a prototype system consists of a mobile robot and motes. In some simple topologies, they showed the adaptive algorithm collects more data than the one that uses constant speed.

Ma and Yang [MY06, MY07] studied the path selection problem. Their objective was to maximize the network lifetime, which is defined as the time until the first node dies (i.e. minimum of the lifetime of all nodes). Speed control was not their focus and they just used a constant speed mobility model. They have presented a heuristic path selection algorithm that is based on divide and conquer approach and finds a near-optimal path for each part of the nodes. When the path of data mule is given, they showed the problem of maximizing the network lifetime is formulated as a flow maximization problem that has a polynomial time algorithm. One limitation is that, they avoid the problem of scheduling issue by assuming either that data communication time is negligible (in [MY06]) or that the data mule stops while communicating with sensor nodes (in [MY07]). However, the former assumption does not apply when data size is big and communication is slow, and the latter results in inefficiency when the data mule can actually communicate while it is moving. Another problem in their algorithm is that it is applicable only to special configurations in which a data mule starts from the left end of the area, travels toward the right end and then comes back to the left end. Hence, for example, it is not clear how to use the algorithm for circular area having the base station in the center.

Xing et al. [XWXJ07, XWJL08] designed path selection algorithms when each node can forward data toward the base station along a routing tree constructed in advance. Their formulation is also similar to TSP and also assumed the existence of forwarding nodes that do not generate data by themselves, in order to make the network connected and enable the construction of routing tree rooted at the base station. Although these assumptions allow the fail-over mechanism that improves the data delivery rate, they also limit the applicability of the technique.

For multiple data mules case, Jea et al. [JSS05] studied the case of multiple data mules move on fixed paths. In their settings, the speed of data mules are also fixed. Instead, their focus was on a distributed coordination scheme for allocating sensor nodes to each data mule to achieve a good distribution of communications load among the data mules.

2.3 Summary

Mobility has been studied in earlier works for the purpose of improving communication performance in wireless networks. In a common communication pattern where there are one or more intermediary nodes between source and destination nodes, each of the participants can be mobile. We have classified the mobility model into stationary, random, predictable, and controlled, and reviewed the literature depending on the mobility models for source, destination, and intermediary nodes. Our focus is on data mule approaches, which are expressed as stationary source and destination nodes with controllably mobile intermediaries.

For data mule approaches we have reviewed the literature more in depth. Table 2.2 compares these approaches in terms of addressed problems, assumed mobility model of data mules, communication model, and so on. Note that, since each work has different objective and focus, this cannot serve as a fair comparison of the overall quality of work. Rather, our intention is to analyze the variety of application models and assumptions that these works have used. Based on this, our problem framework presented in the next chapter is general enough to incorporate most of these varieties, as shown in the bottom of the table.

Table 2.2: Summary of related work on data mule approaches

	Problem		Mobility model (Speed change)	Multihop3)	#DM	Communication model
	Path1)	Speed2)				
Zhao and Ammar [ZA03]	Yes	Yes	Variable	No	Single	Fixed
Kansal et al. [KSJ+04]	No	Yes	Variable	Yes	Single	No assumption
Somasundara et al. [SRS04]	Yes	No	(Given constant)	No	Single	Zero
Ma and Yang [MY06, MY07]	Yes	No	(Given constant)	Yes	Single	Fixed
Xing et al. [XWXJ07, XWJL08]	Yes	No	(Given constant)	Yes	Single	Zero
Zhao et al. [ZAZ05]	Yes	No	(Given constant)	No	Multiple	Zero
Somasundara et al. [SRS07]	Yes	No	(Given constant)	No	Multiple	Zero
Jea et al. [JSS05]	No	No	(Given constant)	Yes	Multiple	Fixed
This dissertation	Yes (Ch. 6)	Yes (Ch. 4, 5)	Constant (Ch. 4), Variable (Ch. 4), Acceleration-constrained (Ch. 5)	Yes (Ch. 7)	Single (Ch. 4-6), Multiple (Ch. 8)	Fixed (Ch. 4-8), Partially-known (Ch. 9)

Notes:

- 1) "Path": Determines the path of data mule
- 2) "Speed": Determines the speed of data mule
- 3) "Multihop": Considers the combined approach of data mule and multihop forwarding

Chapter 3

Problem Framework

This chapter discusses how we formulate the motion planning problem of data mules. We first identify a common application model for data collection using data mules and introduce the Data Mule Scheduling (DMS) as the problem framework that is sufficiently general.

3.1 Need for a Problem Framework

Optimizing the motion of data mules is an important problem for data mule approaches to be useful. However, this is a hard problem since there are a large degree of freedom in choosing the path and speed of the data mules while satisfying both spatial and temporal constraints on communications and data mules' movements. The hardness is also depicted by a simple example with zero communication range, where the problem essentially becomes equivalent to the traveling salesman problem, which is NP-hard.

Due to the hardness, previous studies have simplified the problem in various ways, as we have summarized in Chapter 2. As a result, the proposed algorithms often require specific settings such as the zero communication range (i.e., data mule needs to go to the exact node location to communicate with it) and the constant speed movement of data mule. These assumptions narrow the range of scenarios that these algorithms can be applied.

On the other hand, actual application scenarios such as [VKR$^+$05] and [TMF$^+$07] often have large variety of settings. For example, a data mule can be an underwater vehicle, a ground vehicle like Packbot, and a helicopter. Similarly, sensor nodes can be

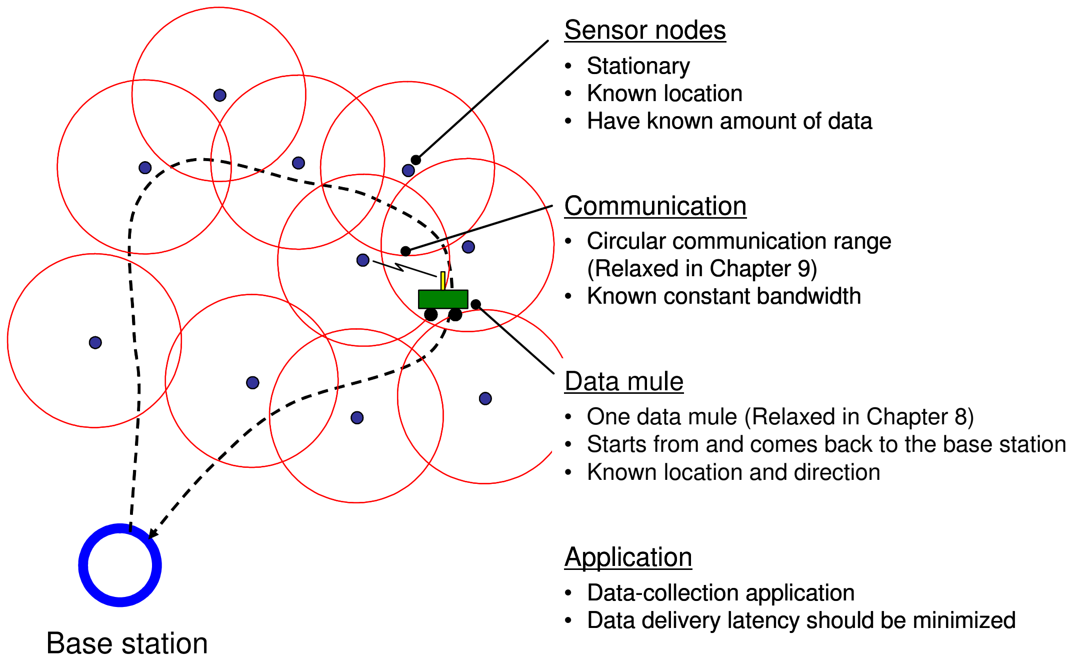

Sensor nodes
• Stationary
• Known location
• Have known amount of data
Communication
• Circular communication range (Relaxed in Chapter 9)
• Known constant bandwidth
Data mule
• One data mule (Relaxed in Chapter 8)
• Starts from and comes back to the base station
• Known location and direction
Application
• Data-collection application
• Data delivery latency should be minimized
Base station

Figure 3.1: Application model and assumptions

Berkeley Motes or RFID-based ones, and communication can be via 802.15.4, optical and acoustic signals.

From these observations, instead of presenting a single problem formulation, we reached a conclusion that we need a problem framework that is general enough to capture the problem space. In such a problem framework, we provide multiple different problem formulations appropriate for each application scenario.

3.2 Application Model and Assumptions

Here we describe the application model for which we design a problem framework and also the assumptions. Figure 3.1 shows the summary. The model is simple and general enough to apply to many of the data-collection application scenarios.

3.2.1 Application Model

Application

- Data-collection application. The objective is to move the data from each sensor node to the base station. As we have discussed in Chapter 1, this is a generic form of majority of sensor network applications.
- Data delivery latency should be minimized. Data delivery latency is defined as the latency from the time data is generated at a sensor node until the time it is delivered to the base station. This is a common requirement for sensor network applications, though the priority may differ. In the structural health monitoring application presented in Section 2.2.1, the priority is high since the safety of the structure needs to be verified as soon as possible after an earthquake. In environmental monitoring applications, the requirement for the minimum data delivery latency may not be very strong from the applications' perspectives. However, the situation is different when a data mule can be active only for a short time due to the limited fuel, which is often the case. To collect the data from all the sensor nodes in the limited amount of time, we need to optimize the motion of the data mule.

Sensor Nodes

- Stationary sensor nodes. Nodes do not move from their original locations. This is a common assumption for sensor network applications and reasonable for majority of applications.
- Sensor nodes have known amount of data (non-periodic case) or generate data at known rate (periodic case).

Data Mule

- There is only one data mule. This assumption is relaxed in Chapter 8 when we consider the multiple data mules case.
- Data mule starts from and comes back to the base station. The base station serves as a gateway to other hosts and thus the most appropriate place to deposit the collected data. It is also reasonable to assume that we can refuel the data mule

at the base station when the data mule needs to collect data periodically, as we consider later.

3.2.2 Assumptions

- Nodes are stationary and their locations are known. Sometimes locations are known by the geometry of the structure to be monitored, or through the use of the sensor network topologies. In other cases, nodes may be localized using various ranging techniques including GPS. The actual process of localization – the participants and the domain where it is done – are out of the scope of this dissertation. For more details on the topic, refer to the survey by Mao et al. [MFA07].
- Known location and direction (data mule). Similarly as the sensor nodes, the data mule knows its current location and the direction it is heading in.
- Circular communication range with known radius. Wireless communication range is circular and the radius is a constant value. Communication within the range is always successful. This assumption is relaxed in Chapter 9 when we discuss the case of partially-known communication range.
- The bandwidth of communication between a node and the data mule is constant. This also applies to the communications between nodes when we discuss a combination with multihop forwarding in Chapter 7.

3.3 Data Mule Scheduling (DMS) Problem

In this section we present the data mule scheduling (DMS) problem framework for optimizing the motion of data mule. As we have discussed earlier, motion planning of data mule is a hard problem. To deal with that, in the DMS problem framework, we decompose the problem into the following four subproblems as shown in Figure 3.2:

1. Forwarding: how each node sends the data to neighboring nodes
2. Path selection: which trajectory the data mule follows
3. Speed control: how the data mule changes the speed while moving along the path
4. Job scheduling: from which sensor the data mule collects data at each time point

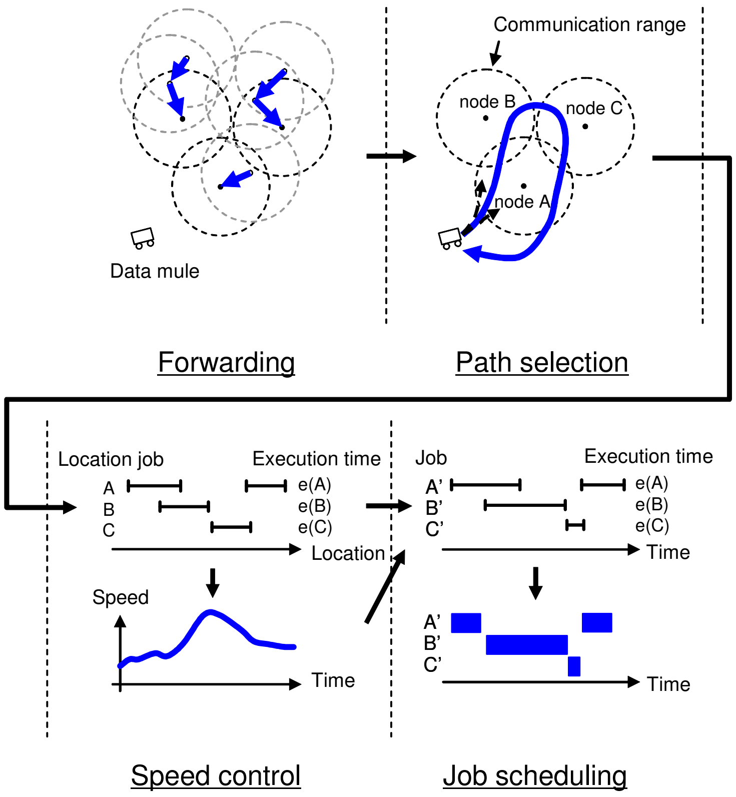


Figure 3.2: Subproblems of the Data Mule Scheduling (DMS) problem

Forwarding subproblem is for a combined approach of data mule and multihop forwarding. The problem to determine the amount that each node forwards its data to neighboring nodes within the communication range. For each node, the difference between incoming data (including generated data at the node) and outgoing data is the amount that needs to be collected by the data mule. An important point is that, when the difference is zero, the data mule does not need to collect any data from this node.

Path selection subproblem is to determine the trajectory of the data mule in the sensor field. To collect data from each particular sensor, the data mule needs to go within the sensor's communication range at least once. Depending on the capability of data mule, there may be some constraints on path selection, such as minimum turning radius.

Speed control subproblem is to determine how the data mule changes its speed along the chosen path. The data mule needs to change the speed so that it stays within each sensor's communication range long enough to collect all the data from it.

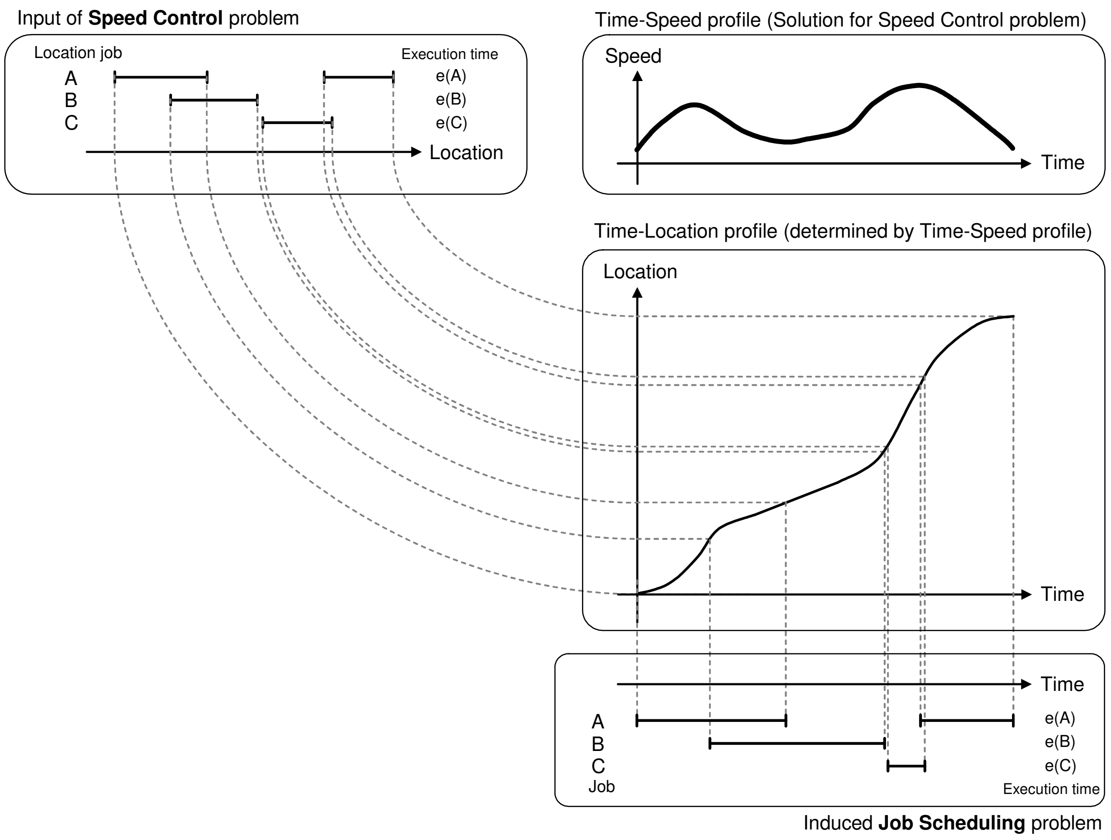

Input of Speed Control problem
Location job
Execution time
A
B
C
e(A)
e(B)
e(C)
Location
Time-Speed profile (Solution for Speed Control problem)
Speed
Time
Time-Location profile (determined by Time-Speed profile)
Location
Time
Time
A
B
C
Job
e(A)
e(B)
e(C)
Execution time
Induced Job Scheduling problem

Figure 3.3: Relation between speed control and job scheduling subproblems

The final subproblem is job scheduling. Once the time-speed profile is determined, we get a mapping from each location to a time point, as shown in Figure 3.3. Thus we get a job scheduling problem by regarding data collection from each sensor as a job. Each job has one or more intervals in which it can be executed. Job scheduling subproblem is to determine the allocation of time to jobs so that all jobs can be completed before their deadlines.

3.4 Discussions

One of the biggest benefits of the DMS problem framework is that it allows us a systematic treatment of the motion planning problem of a data mule. The DMS problem enables us to identify the cases of the problem that we can optimally solve, as well as the hard cases that we explore approximation and heuristic algorithms.

Another benefit is that the DMS problem is general and can be used to express several earlier problems in the area. For instance, the assumption of zero communication range (as in [SRS04, SRS07]) is easily expressed by setting the communication range to zero in the path selection subproblem. The constant speed assumption (as in [MY06, MY07, XWXJ07]) and variable speed assumption (as in [ZA03, KSJ+04]) are handled in the speed control subproblem. Further, the framework can easily accommodate new settings such as acceleration constrained case, which we discuss later in Chapter 5, and also different ways of problem formulation, as we discuss in Chapter 6.

In contrast to these benefits, one of the limitations of the DMS problem is that there is a lost opportunity of further improvements by dividing the original problem into mostly independent subproblems. For example, we solve the two dimensional case of the problem by solving the path selection subproblem first and reducing it to the one dimensional problem. Nevertheless, the DMS problem framework is useful because it facilitates fundamental understanding of the problem in many ways by providing a unified treatment of the 1-D and 2-D problems. Furthermore, it allows multiple different formulations of each subproblems, and thereby provides a variety of choices in balancing the optimality of solutions and tractability of the problem.

3.5 Summary

In this chapter we have presented the Data Mule Scheduling (DMS) problem as a problem framework for motion planning of a data mule. For the framework to be broadly applicable, we have designed a simple application model that captures the common features of data-collection applications that use controlled mobility. The key idea for the DMS problem framework is that we split the problem into the subproblems of forwarding, path selection, speed control, and job scheduling. Each subproblem is much simpler than the original joint optimization problem and thereby enables a systematic treatment of the motion planning problem. Furthermore, the DMS problem framework is general enough to be capable of expressing the assumptions used in the previous literature.

Chapter 4

Motion Planning on Fixed Path: Basic Cases

A basic case of the DMS problem framework is when the path of the data mule is fixed. This is called the 1-D DMS problem and requires determination of the speed change and the schedule of communication with the sensor nodes. We consider three different mobility models of the data mule: Constant speed, Variable speed, and Generalized. We call the first two models the basic cases, which are the focus of this chapter. Generalized model is discussed in the next chapter. We formally define the 1-D DMS problem for each mobility model and present either optimal algorithms or non-existence proofs of such algorithms. Later in the chapter we discuss the case of periodic data generation and also the connections with speed scaling problems.

4.1 1-D DMS Problem

4.1.1 Terminology, Definitions, and Assumptions

A *job* τ_i has an execution time e_i and a set $\mathcal{I}_i$ of feasible intervals. A *feasible interval* $I \in \mathcal{I}_i$ is a time interval $[r(I), d(I)]$, where $r(I)$ is a *release time* and $d(I)$ is a *deadline*. A job can be executed only within its feasible intervals. A *simple job* is a job with one feasible interval, whereas a *general job* can have multiple feasible intervals. For instance, in Figure 4.1(a), jobs A' and B' are simple jobs and jobs C' and D' are general jobs.

Similarly for the speed control subproblem, a *location job* τ_i has an execution

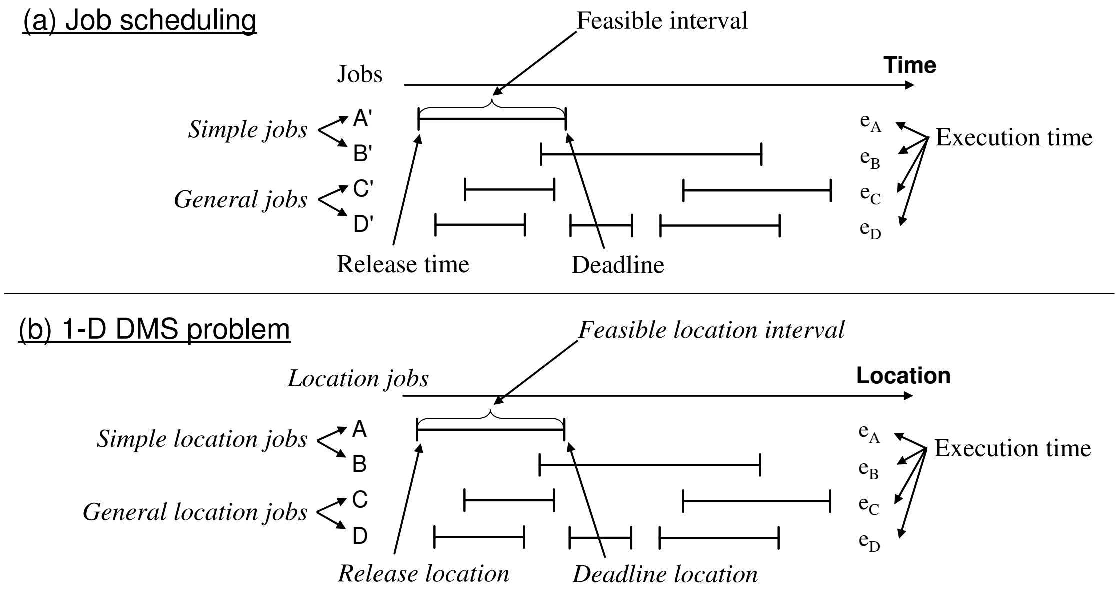


Figure 4.1: Terminology for job scheduling and the 1-D DMS problem: Newly introduced terms are shown in italic.

time e_i and a set $\mathcal{I}_i$ of feasible location intervals. A *feasible location interval* $I \in \mathcal{I}_i$ is a location interval $[r(I), d(I)]$, where $r(I)$ is a *release location* and $d(I)$ is a *deadline location*. A location job can be executed only within its feasible location intervals. A *simple location job* is a location job with one feasible location interval, whereas a *general location job* can have multiple feasible location intervals. In Figure 4.1(b), location jobs A and B are simple location jobs and location jobs C and D are general location jobs. We may omit "location" unless it is ambiguous.

An important observation is that, as shown in Figure 3.3, a location job is mapped to a job when the time-speed profile is given. Furthermore, the reason why we consider general (location) jobs is that the path may intersect with the communication range of a node multiple times (e.g., node A in Figure 3.2).

For an interval $I = [r, d]$ (also for a location interval), $|I|$ denotes the length $d-r$. We also define containment as follows: $I \subseteq I'$ if and only if $r' \leq r$ and $d \leq d'$ where $I' = [r', d']$.

Following the definitions in [Liu00], we use the term *offline scheduling algorithm* when a scheduling algorithm computes a schedule based on the complete knowledge of all the jobs for all times. On the other hand, an *online scheduling algorithm* computes

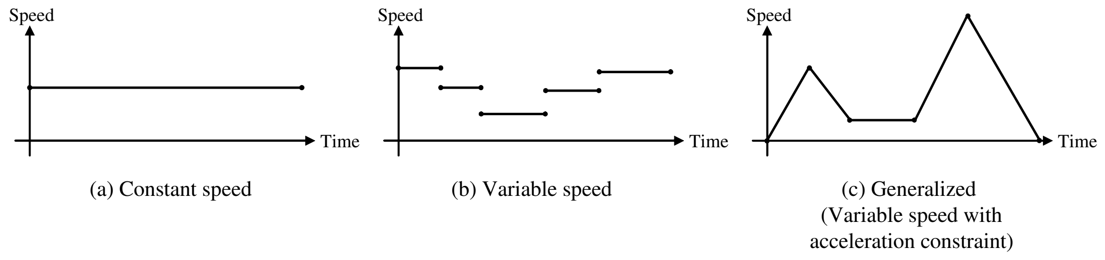


Figure 4.2: Mobility models of data mule

a schedule without knowledge about the jobs that will be released in the future. In this case, the parameters of each job become known to the online scheduler when the job is first released.

With slight extensions for general jobs, we also define validity, feasibility, and optimality based on [Liu00]. A schedule is *valid* when it satisfies 1) every processor executes at most one job at any time; 2) no job is scheduled other than in one of its feasible intervals; 3) total amount of processor time assigned to every job is equal to its execution time. A valid schedule is *feasible* when every job completes by the deadline of its final feasible interval. Finally, a scheduling algorithm is *optimal* when it always finds a feasible schedule if the given set of jobs has a feasible feasible.

The definitions of offline/online scheduling algorithms, validity, feasibility, and optimality are extended for the 1-D DMS problem as well.

The assumptions are as follows. Each sensor node is stationary. Communication range and execution time are known. Communication is always successful in the communication range. All location jobs are preemptible without any cost incurred and can be executed over multiple feasible location intervals. There is no dependency among the location jobs. Data mule can communicate with one node at a time. Depending on the dynamics constraint, data mule may have constraints on the maximum speed and maximum acceleration.

4.1.2 Mobility Models

We consider the following three mobility models with different dynamics constraints:

Constant speed $v(t) = v_0$ for some constant v_0

Variable speed $v_{min} \le v(t) \le v_{max}$ for given v_{min}, v_{max} $(0 \le v_{min} \le v_{max})$

Generalized $v(0) = v(T) = 0$, $0 \le v(t) \le v_{max}$, $|dv(t)/dt| \le a_{max}$ for given $v_{max} > 0$, $a_{max} > 0$

Figure 4.2 shows the difference of these models.

Constant speed model represents the case where the data mule cannot change the speed after it starts to move. Variable speed model represents the case where the data mule can instantaneously change the speed within a given speed range. It also captures the case where the data mule stops at each node for communication. Generalized model is with an acceleration constraint. In this model, the data mule can change the speed, but the rate of change is within a given maximum absolute acceleration. This model is called "generalized" in the sense that the former two cases are roughly the special cases of this case when $a_{max} = 0$ and $a_{max} = +\infty$, respectively.

The constant speed and variable speed models apply to ground vehicles such as Packbot[1], which is commonly used as a data mule in actual deployments. The generalized model captures mobility more precisely and is most appropriate when we cannot ignore the inertia, for example in case that a helicopter is used as a data mule as in [TMF+07].

4.1.3 Problem Definition

An instance of the 1-D DMS problem is $(L, \mathcal{J})$, where

- $[0, L]$: total travel interval of the data mule on the location axis
- $\mathcal{J}$: set of location jobs; i-th location job τ_i is characterized by
  - $\mathcal{I}_i$: set of feasible location intervals
  - e_i: execution time

A solution to the problem is a pair of time-speed profile $v(t)$ and a job schedule. Let T denote the total travel time. Then the constraints on the motion of data mule are $\int_0^T v(t) = L$ and the dynamics constraints according to each mobility model. After $v(t)$ is determined, we can define a function $f(x)$ that maps location x to time t. Using $f(x)$, we obtain a job scheduling problem, which we call an *induced job scheduling problem.* This mapping is shown in Figure 3.3. We need to determine $v(t)$ such that the induced

[1]`http://www.irobot.com/`

job scheduling problem has a feasible schedule. The objective of the 1-D DMS problem is to find a solution that minimizes the travel time T.

4.2 Related Job Scheduling Problems

The 1-D DMS problem is transformed to a job scheduling problem once we determine a speed control plan of the data mule. Here we discuss some related issues in job scheduling problems.

We extend the definition of normal scheduling problem to general jobs, i.e., jobs with multiple feasible intervals. Preemptive Scheduling for General Jobs is defined as follows: Given a set $\mathcal{J}$ of jobs, for each job $\tau \in \mathcal{J}$, an execution time $e(\tau)$ and a set $\mathcal{I}(\tau)$ of feasible intervals, for each feasible interval $I \in \mathcal{I}(\tau)$, a release time $r(I)$ and a deadline $d(I)$, is there a feasible schedule for $\mathcal{J}$?

4.2.1 Offline Scheduling Algorithms

For the simple jobs case, we can use the Earliest Due Date (EDD) algorithm as an offline scheduling algorithm and it is optimal [Jac55, LL73, SSNB95]. In the EDD algorithm, a job with the earliest deadline among all available jobs is executed at any time slice.

For the general jobs case, however, the EDD algorithm is not well-defined, since each job has multiple deadlines. Instead, we can design an optimal offline algorithm by linear programming, which can be solved efficiently by using any LP solver. Without losing generality, assume the earliest release time of all jobs is at time 0 and the latest deadline is at time T. We split the time interval $[0, T]$ into $(2m+1)$ intervals $[t_0(=0), t_1]$, $[t_1, t_2]$, ..., $[t_{2m}, t_{2m+1}(=T)]$, where $t_i \in P_r \cup P_d$, $t_i \le t_{i+1}$ and P_r, P_d are the set of release time and deadline for all the jobs, respectively. For every job $\tau \in \mathcal{J}$, consider variables $p_0(\tau), ..., p_{2m}(\tau)$, in which $p_i(\tau)$ represents the time allocated to job τ during the interval $[t_i, t_{i+1}]$.

We construct a linear programming problem to find a feasible solution of $p_i(\tau)$ satisfying following constraints:

- $0 \le \forall i \le 2m, \forall \tau \in \mathcal{J}$, $p_i(\tau) \ge 0$,
- (Feasible interval) $\forall \tau \in \mathcal{J}$, if $[t_i, t_{i+1}] \notin \mathcal{I}(\tau)$, $p_i(\tau) = 0$,

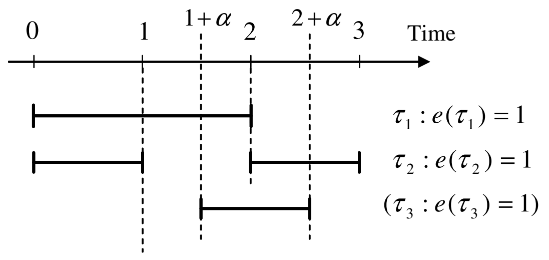


Figure 4.3: Counterexample for showing non-existence of optimal online scheduling algorithm for Preemptive Scheduling for General Jobs.

- (Job completion) $\forall \tau \in \mathcal{J}$, $\sum_{i=0}^{2m} p_i(\tau) = e(\tau)$,
- (Processor demand) $0 \leq \forall i \leq 2m$, $\sum_{\tau \in \mathcal{J}} p_i(\tau) \leq t_{i+1} - t_i$.

Note that t_i's are given constants and thus this is a linear programming problem. After obtaining a feasible solution, we divide each interval $[t_{i-1}, t_i]$ to each job τ such that τ is allocated for time $p_i(\tau)$ within the interval. The order of jobs within each interval is arbitrary.

4.2.2 Non-Existence of Optimal Online Scheduling Algorithm

To be more practical, it would be preferable if we have an online scheduling algorithm. An online scheduling algorithm determines the schedule solely based on the information of the jobs that are already released. We assume the information of a job, including its execution time and all feasible intervals, becomes available to the scheduler when the job is first released. For simple jobs, the EDD algorithm is an optimal online scheduling algorithm. For general jobs, the following theorem gives a negative result:

Theorem 4.2.1. *There is no optimal online scheduling algorithm for Preemptive Scheduling for General Jobs.*

Proof. Assume there exists an optimal online scheduling algorithm. We give an example set of general jobs to show that, for any schedule until a certain time, an adversary can issue a job that leads to a scheduling failure, yet the set of jobs as a whole remains to have a feasible schedule.

Figure 4.3 is a counterexample consisting of three general jobs, although job τ_2 is the only one having multiple feasible intervals. Suppose an optimal online scheduling algorithm allocates time p and q $(0 \leq p, q \leq 1, p + q \leq 1)$ to jobs τ_1 and τ_2, respectively, in the interval $[0, 1]$. We assume $q = 1 - p$ since the algorithm is optimal. Note that, at time 1, the algorithm has no information about job τ_3, which is released later. Assume job τ_3 has a feasible interval $[1 + \alpha, 2 + \alpha]$ and the execution time is one. Since the length of the feasible interval equals to the execution time, the interval $[1 + \alpha, 2 + \alpha]$ needs to be allocated solely to job τ_3 to finish it. When $0 \leq p < 1$, the adversary chooses $\alpha = 1-p-\varepsilon$, where ε is a small constant satisfying $\varepsilon > 0,\ p+\varepsilon < 1$. Then, job τ_1 cannot be finished because it can be allocated up to $1 - p - \varepsilon$ within the interval $[1, 2]$ and the total allocation to job τ_1 is $1 - \varepsilon$, which is smaller than its execution time. When $p = 1$, the adversary chooses $\alpha = \varepsilon$ and then job τ_2 cannot be finished for a similar reason. Clearly, the set of jobs has a feasible schedule: allocating $[0, \alpha]$ to job τ_2, $[\alpha, 1 + \alpha]$ to job τ_1 to finish, $[1 + \alpha, 2 + \alpha]$ to job τ_3 to finish, and $[2 + \alpha, 3]$ to job τ_2 to finish. This is a contradiction to the assumption that the algorithm is optimal. $\square$

4.3 Constant Speed 1-D DMS Problem

One of the simplest forms of the 1-D DMS problem is the constant speed case, where a data mule moves at a constant speed from the start to the destination. Constant Speed 1-D DMS problem is defined by adding the constant movement constraint to the 1-D DMS problem. Specifically, the question for the decision version is to ask the feasibility when the data mule moves at the constant speed $v_0 = L/T$, where L is the total travel length on the location axis and T is the total travel time. The optimization version of the problem is defined by changing the question to finding the minimum T, or equivalently to finding the maximum v_0. In this section, we give algorithms for the optimization version of the problem.

When we assume the data mule cannot change the speed once it starts to move, it is easy to show that there is no optimal online scheduling algorithm. Suppose, for the sake of contradiction, such an algorithm exists and determines the optimal speed v_0 for a certain set of location jobs. Then, since the algorithm by definition does not know about the jobs released (for the first time) in the future, it cannot complete a job released in the future that has an execution time e and a feasible location interval of length $(v_0 - \varepsilon)e$ ($\varepsilon > 0$), which contradicts the assumption.

In the rest of the section, we present an optimal offline algorithm for simple location jobs and a linear program formulation for general location jobs.

4.3.1 Simple Location Jobs

When each location job has one feasible location interval, we can use a processor-demand based feasibility testing to find the optimum (i.e., maximum) v_0 such that all location jobs can be completed. Processor demand g in interval $[t_1, t_2]$ is the sum of the execution time of the tasks whose feasible interval is completely contained in the interval, and is defined as

$$g(t_1, t_2) \quad = \quad \sum_{\substack{\tau : \tau \in \mathcal{J}, \\ I(\tau) \in [t_1, t_2]}} e(\tau).$$

In real time scheduling, the following theorem holds for periodic tasks with arbitrary relative deadline (i.e., relative deadline of each task can be smaller than its period):

Theorem 4.3.1 (Baruah et al. [BHR93]). *Let $\tau = \{T_1, ..., T_n\}$ be a task system. τ is not feasible iff there exist natural numbers $t_1 < t_2$ such that $g(t_1, t_2) > t_2 - t_1$*

Using the theorem, we can show that testing at each release time and deadline is sufficient to guarantee the feasibility. In other words,

Theorem 4.3.2. *Task system is feasible iff $g(t_1', t_2') \leq t_2' - t_1'$ for any $t_1' \in \{r_i\}, t_2' \in \{d_i\}$ satisfying $t_1' < t_2'$.*

To prove Theorem 4.3.2, we first show that, for any arbitrary interval, there exists a pair of release time and deadline that has the same processor demand, as per the following lemma:

Lemma 4.3.3. *For any t_1, t_2 satisfying $t_1 < t_2$ and $g(t_1, t_2) > 0$, there exist $t_1' \in \{r_i\}$ and $t_2' \in \{d_i\}$ such that $t_1 \leq t_1' < t_2' \leq t_2,\ g(t_1, t_2) = g(t_1', t_2')$.*

Proof. Choose t_1', t_2' as follows:

$$\begin{aligned} t_1' &= \min_{i : \tau_i \in T, r_i \geq t_1} \{r_i\} \\ t_2' &= \max_{i : \tau_i \in T, d_i \leq t_2} \{d_i\} \end{aligned}$$

Since there is no task released in interval $[t_1, t_1')$, $g(t_1, t_2) = g(t_1', t_2)$. Similarly, since there is no task having a deadline in interval $(t_2', t_2]$, $g(t_1', t_2) = g(t_1', t_2')$. Therefore, $g(t_1, t_2) = g(t_1', t_2')$ for these t_1', t_2'. Further, since $g(t_1, t_2) > 0$, $[t_1, t_2]$ contains at least one task, and thus $t_1 \leq t_1' < t_2' \leq t_2$. $\square$

Lemma 4.3.4. *$g(t_1, t_2) \le t_2 - t_1$ for any t_1, t_2 satisfying $t_1 < t_2$ if and only if $g(t'_1, t'_2) \le t'_2 - t'_1$ for any $t'_1 \in \{r_i\}, t'_2 \in \{d_i\}$ satisfying $t'_1 < t'_2$.*

Proof. "Only if" part is obvious, and we show "if" part with a proof by contrapositive. We first assume $\exists t_1 < t_2,\ g(t_1, t_2) > t_2 - t_1$ and prove $\exists t'_1 \in \{r_i\}, \exists t'_2 \in \{d_i\}, (t'_1 \le t'_2) \wedge (g(t'_1, t'_2) > t'_2 - t'_1)$. By Lemma 4.3.3, there exist $t'_1 \in \{r_i\}$ and $t'_2 \in \{d_i\}$ such that $g(t'_1, t'_2) = g(t_1, t_2) > t_2 - t_1$. Choose $t'_1 = \min_{i:r_i \ge t_1}\{r_i\}$ and $t'_2 = \max_{i:d_i \le t_1}\{d_i\}$. Since $t_2 - t_1 > 0$, $g(t_1, t_2) > 0$ and there is at least one task contained in interval $[t_1, t_2]$. Thus $t'_2 - t'_1 > 0$ and $t_2 - t_1 \ge t'_2 - t'_1$. Therefore, $\exists t'_1 \in \{r_i\}, \exists t'_2 \in \{d_i\}, (t'_1 \le t'_2) \wedge (g(t'_1, t'_2) > t'_2 - t'_1)$. □

Theorem 4.3.2 follows from Theorem 4.3.1 and Lemma 4.3.4.

This feasibility testing is for periodic task system in real time scheduling, but we can apply it for our case. Specifically, we can determine the speed for each location interval so that the processor demand for the interval is at most the time that the data mule stays in the interval.

When each location job has one feasible location interval, the following simple algorithm in Algorithm 1 finds the maximum possible v_0 such that all location jobs can be completed. It applies the processor-demand based feasibility test (Theorem 4.3.2) for all possible pairs of a release location and a deadline location.

Algorithm 1 FIND-MIN-MAXSPEED

```
1: for each location interval $I = [r(\tau'), d(\tau'')]$ s.t. $\tau', \tau'' \in \mathcal{J}$, $r(\tau') \le d(\tau'')$ do
2:     $d = \sum_{\substack{\tau:\tau\in\mathcal{J},\\ \mathcal{I}(\tau)\in I}} e(\tau)$                ▷ Processor demand for $I$
3:     $u[I] \leftarrow \frac{|I|}{d}$                ▷ Maximum speed allowed for $I$
4: end for
5: return $\min_I u[I]$
```

This algorithm runs in $O(n^3)$ time where n is the number of location jobs in a naive implementation, but we can improve it to $O(n^2)$ by computing the processor demand incrementally. Specifically, for each starting location, by having a list of jobs sorted by their deadline locations, we can incrementally extend the interval and calculate the processor demand in $O(1)$ time. Then it takes $O(n)$ time for each starting location, and since there are at most n starting locations, it takes $O(n^2)$ as a whole.

This algorithm is correct and optimal for the following reasons. For correctness, Theorem 4.3.2 guarantees the feasibility iff, for all possible pairs of release time and deadline, the processor demand for the interval is equal to or less than the length of the interval. In the algorithm, this condition is satisfied since we choose the minimum of maximum possible speed for all pairs of release and deadline locations. Optimality is shown from the same argument: as the processor demand condition above is both necessary and sufficient condition for feasibility, v_0 chosen by the algorithm is the maximum possible speed such that the corresponding set of jobs remains feasible.

4.3.2 General Location Jobs

For the general location jobs case where each job may have multiple feasible location intervals, we treat execution time for each feasible location interval as a parameter and formulate the problem as a linear program. We solve the optimization version of the problem by regarding the speed of data mule v_0 as a variable to maximize.

The formulation is almost the same as the one for Preemptive Scheduling for General Jobs (in Section 4.2.1), except that now we map location to time. We split the total travel interval $[0, L]$ into $(2m+1)$ location intervals $[l_0(=0), l_1]$, $[l_1, l_2]$, ..., $[l_{2m}, l_{2m+1}(=L)]$ $(l_i \leq l_{i+1})$, where m is the number of feasible location intervals of all location jobs, and each l_i is either a release location or a deadline location. For each location job $\tau \in \mathcal{J}$, we consider variables $p_0(\tau), ..., p_{2m}(\tau)$, in which $p_i(\tau)$ represents the time allocated to job τ within the location interval $[l_i, l_{i+1}]$. Then the objective of linear program is to maximize v_0, and the constraints are the same as in Section 4.2.1 except that the processor demand constraint is replaced by

- (Processor demand) $0 \leq \forall i \leq 2m$, $\sum_{\tau \in \mathcal{J}} p_i(\tau) \leq (l_{i+1} - l_i)/v_0$.

Note that this constraint becomes a linear constraint by introducing a new variable $u_0 = 1/v_0$ and changing the objective to the minimization of u_0.

4.4 Variable Speed 1-D DMS Problem

Another simple form of the 1-D DMS problem is the variable speed case, where a data mule can change the speed. In this section, we discuss the case in which speed changes can be instantaneous. Variable Speed 1-D DMS problem is defined by adding the speed range $[v_{min}, v_{max}]$ $(0 \leq v_{min} \leq v_{max})$ to the input of 1-D DMS and allowing the speed change within the range[2]. The optimization version is defined similarly as the constant speed case. Without loss of generality, we can restrict the speed control plan to be a piecewise constant function. We can assume this because the feasibility of the corresponding job scheduling problem is solely determined by how the release and deadline locations are mapped onto the time axis.

4.4.1 Simple Location Jobs

For simple location jobs case, we have the following optimal offline algorithm, which is based on Yao et al.'s algorithm [YDS95] for processor speed scaling. The algorithm is based on processor-demand analysis and uses Find-Min-MaxSpeed internally.

Algorithm 2 Sequential-Find-Min-MaxSpeed

```
1: loop
2:     $v_c \leftarrow$ Find-Min-MaxSpeed
3:     if $v_c < v_{min}$ then return INFEASIBLE
4:     else if $v_c > v_{max}$ then
5:         Set $v_{max}$ for all remaining intervals and finish
6:     else
7:         Set $v_c$ for the current tight interval
8:         Remove jobs within the tight interval
9:         "Compress" remaining jobs
10:    end if
11: end loop
```

Here is how this algorithm works. For the set of location jobs, Find-Min-MaxSpeed finds a tight interval and the corresponding speed v_c. It is the minimum speed that makes the processor demand for each location interval equal or less than the time allocated to it. In other words, if the data mule moves at the speed faster than v_c, there is at least one interval in which the allotted time is less than the processor demand (thus violates the feasibility). Therefore, if $v_c < v_{min}$ (Line 3), it is infeasible. "Compress" (Line 9) is an operation used in [YDS95], which is to remove the tight interval from the feasible intervals of all jobs and to connect the remaining two intervals together

[2]Without a speed range, the problem is trivial because a data mule can alternate moving at infinite speed and stopping to execute a job.

to construct a new set of jobs. In each iteration, v_c is nondecreasing. This is shown by the same reasoning in [YDS95]. When v_c reaches v_{max}, we cannot increase the speed of any remaining location intervals. Thus we set the speed to v_{max} for all these intervals (Line 5).

This algorithm runs in $O(n^3)$ time, since each iteration takes $O(n^2)$ time by using the improved implementation of Find-Min-MaxSpeed and at least one job is removed at each iteration.

As for an online scheduling algorithm, we need to consider $v_{min} = 0$ and $v_{min} > 0$ cases separately. When $v_{min} = 0$, the following algorithm is an optimal online algorithm that minimizes the total travel time. In the algorithm, EDD-with-Stop, the data mule moves at v_{max} while executing a job having the earliest deadline, just in the same way as ordinary Earliest Due Date algorithm. However, when a job is not completed at its deadline, the data mule stops until the job is completed and moves at v_{max} again.

We have a following theorem about the optimality of EDD-with-Stop.

Theorem 4.4.1. EDD-with-Stop *is optimal for Variable Speed 1-D DMS problem for simple location jobs when* $v_{min} = 0$

Proof. Every feasible schedule that uses the speed other than 0 and v_{max} can be converted to one that only uses 0 and v_{max}. Thus, for a feasible schedule, the total time duration to use the speed 0 (i.e., stop) is minimized if and only if the schedule is optimal. Further, for a feasible schedule that does not have idle time while stopping, the idle time while moving at v_{max} is minimized if and only if the schedule is optimal.

We consider another job scheduling problem in which we want to maximize the allocated time, or equivalently to minimize the idle time. A job cannot be executed after its deadline, but non-standard assumptions are that the system may be overloaded and that partial job execution counts in this problem. We claim the following algorithm similar to the EDD algorithm is optimal for the problem:

> Algorithm $\mathcal{A}$: At any time, execute a job with the earliest deadline among executable jobs.

Note that this algorithm is identical to the EDD algorithm when the system is underloaded. We show this algorithm minimizes the idle time by converting an optimal schedule to it. Let $A_{\mathcal{A}}$ and A_{opt} denote the allocation by the algorithm and the optimal schedule, respectively. Allocation during time interval $[t_a, t_b]$ is denoted as $A(t_a, t_b)$. We

Algorithm 3 EDD-with-Stop

```
Init $\mathcal{J}_P$: set of "pending" location jobs, i.e., the location jobs that are currently executable and not finished yet; initialize with $\emptyset$
    $v$: data mule's speed; initialize with $v_{max}$
    $a(\tau)$: time allocated to job $\tau$; initialize with 0
    $x_c$: current location; initialize with 0

On $\exists\tau \in \mathcal{J}_P,\ x_c = d(\tau)$        ▷ When a job is unfinished at its deadline location
 1: $\mathcal{J}_u \leftarrow \{\tau | \tau \in \mathcal{J}_P, d(\tau) = x_c\}$        ▷ Set of jobs that needs to be finished here
 2: $v \leftarrow 0$        ▷ Data mule stops
 3: Complete each job in $\mathcal{J}_u$
 4: $\mathcal{J}_P \leftarrow \mathcal{J}_P \setminus \mathcal{J}_u$
 5: $v \leftarrow v_{max}$        ▷ Move at $v_{max}$ again
 6: $\tau_{ed} \leftarrow \arg\min_{\tau\in\mathcal{J}_P} d(\tau)$        ▷ Job with the earliest deadline location
 7: Execute $\tau_{ed}$

On $\exists\tau \in \mathcal{J},\ x_c = r(\tau)$        ▷ When a job is released
 8: $\mathcal{J}_P \leftarrow \mathcal{J}_P \cup \{\tau | r(\tau) = x_c\}$
 9: $\tau_{ed} \leftarrow \arg\min_{\tau\in\mathcal{J}_P} d(\tau)$
10: Execute $\tau_{ed}$

On $\exists\tau \in \mathcal{J}_P,\ a(\tau) = e(\tau)$        ▷ When a job is finished
11: $\mathcal{J}_P \leftarrow \mathcal{J}_P \setminus \tau$
12: $\tau_{ed} \leftarrow \arg\min_{\tau\in\mathcal{J}_P} d(\tau)$
13: if $\tau_{ed} \neq \emptyset$ then Execute $\tau_{ed}$
14: end if
```

compare $A_{\mathcal{A}}$ and A_{opt} from the beginning and swap allocations in A_{opt} as follows when they differ:

- Case 1: $A_{opt}(t_a, t_b) = \tau_1$, $A_{\mathcal{A}}(t_a, t_b) = \tau_2$, $\tau_1 \neq \tau_2$
  In this case, there exists a pair (t'_a, t'_b) such that $t'_a \geq t_b$ and $A_{opt}(t'_a, t'_b) = \tau_2$, since the time allocated to τ_2 by the time t_a in A_{opt} is shorter by $(t_b - t_a)$ than that in $A_{\mathcal{A}}$, and thus τ_2 is not finished yet in A_{opt}. Accordingly, we can make a list of pairs $L = \{(t'_a, t'_b)\}$ such that $A_{opt}(t'_a, t'_b) = \tau_2$ and $\sum_{(t'_a,t'_b)\in L}(t'_b - t'_a) = t_b - t_a$. For all pairs (t'_a, t'_b) in L, we swap the allocation and obtain $A_{opt}(t'_a, t'_b) = \tau_1$ and $A_{opt}(t_a, t_b) = \tau_2$, which makes the allocation in (t_a, t_b) identical to $A_{\mathcal{A}}$. It is possible because any t'_b is before the deadline of τ_1, since $t'_b \leq d(\tau_2) \leq d(\tau_1)$ (because of EDD-based allocation in $A_{\mathcal{A}}$).

- Case 2: $A_{opt}(t_a, t_b) = \emptyset$, $A_{\mathcal{A}}(t_a, t_b) = \tau$
  From the same argument as Case 1, job τ is not finished in A_{opt} at t_a, and we can swap the allocation to obtain $A_{opt}(t'_a, t'_b) = \emptyset$ and $A_{opt}(t_a, t_b) = \tau$ for a list of pairs $L = \{(t'_a, t'_b)\}$ such that $t'_a \geq t_b$, $A_{opt}(t'_a, t'_b) = \tau$, and $\sum_{(t'_a,t'_b)\in L}(t'_b - t'_a) = t_b - t_a$.

- Case 3: $A_{opt}(t_a, t_b) = \tau$, $A_{\mathcal{A}}(t_a, t_b) = \emptyset$
  This does not happen for the following reason. Since the allocation up to time t_a is identical, τ is not finished yet in $A_{\mathcal{A}}$ at t_a. However, it is a contradiction, since τ is executable at t_a and the algorithm $\mathcal{A}$ allocates time to a job whenever there are any available jobs.

EDD-WITH-STOP allocates exactly the same way as this algorithm $\mathcal{A}$ when the data mule is moving (at v_{max}), thus minimizes the idle time while moving. Therefore, EDD-WITH-STOP minimizes the total travel time. □

When $v_{min} > 0$, the following theorem states that there is no optimal online scheduling algorithm.

Theorem 4.4.2. *There is no optimal online scheduling algorithm for Variable Speed 1-D DMS for simple location jobs when $v_{min} > 0$.*

Proof. When $v_{min} = v_{max}$, it is equivalent to the constant speed case and we have shown that there is no optimal online scheduling algorithm. Thus we consider the case of $v_{min} < v_{max}$. Figure 4.4 shows the counterexample we use. We set the total travel

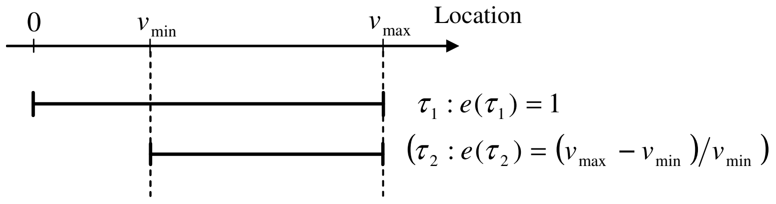


Figure 4.4: Counterexample for showing non-existence of optimal online scheduling algorithm for Variable Speed 1-D DMS for simple location jobs when $v_{min} > 0$.

interval to $[0, v_{max}]$. Let p denote the time the optimal algorithm spent on moving from location 0 to v_{min}. Since the range of speed is $[v_{min}, v_{max}]$ and $v_{min} < v_{max}$, we have $v_{min}/v_{max} < p \leq 1$. Since location job τ_1 is the only available job in this interval, the optimal algorithm spends time p on executing it.

When $p < 1$, an adversary releases location job τ_2, which has feasible location interval $[v_{min}, v_{max}]$ and execution time $(v_{max} - v_{min})/v_{min}$. Since the data mule can spend at most $(v_{max} - v_{min})/v_{min}$ to move from v_{min} to v_{max}, it needs to execute location job τ_2 for the whole time to finish it, and thus it is impossible to finish both τ_1 and τ_2. The set of location jobs is schedulable by the following offline algorithm: moving at v_{min} all the time, first execute τ_1 (finish at location v_{min}) and then τ_2.

When $p = 1$, the adversary does not release location job τ_2. Then the total travel time is at least $1 + (v_{max} - v_{min})/v_{max}$, which is strictly larger than 1. However, it is not optimal, since an optimal offline schedule can reduce the total travel time to 1, by moving at v_{max} all the time and finishing location job τ_1. □

4.4.2 General Location Jobs

For general location jobs, the following theorem states that there is no optimal online scheduling algorithm:

Theorem 4.4.3. *There is no optimal online scheduling algorithm for Variable Speed 1-D DMS for general location jobs.*

Proof. For the example in Figure 4.5, by a similar adversary argument as in the proof of Theorem 4.2.1. □

For general location jobs, we design an offline algorithm by linear programming formulation. The formulation is similar to the one in constant speed case, but we have

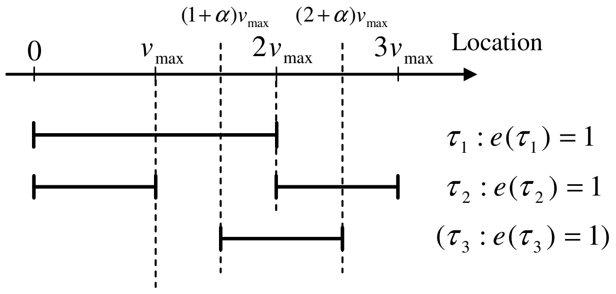


Figure 4.5: Counterexample for showing non-existence of optimal online scheduling algorithm for Variable Speed 1-D DMS for general location jobs.

additional variable z_i for each location interval $[l_i, l_{i+1}]$ to represent the time that the data mule spends in the interval. The linear program is as follows:

Minimize $\sum_{i=0}^{2m} z_i$

Subject to

- $0 \leq \forall i \leq 2m, \forall \tau \in \mathcal{J}$, $p_i(\tau) \geq 0$, $z_i \geq 0$
- (Feasible intervals) $0 \leq \forall i \leq 2m, \forall \tau \in \mathcal{J}$, if $[l_i, l_{i+1}] \notin \mathcal{I}(\tau)$, $p_i(\tau) = 0$
- (Job completion) $\forall \tau \in \mathcal{J}$, $\sum_{i=0}^{2m} p_i(\tau) = e(\tau)$
- (Processor demand) $0 \leq \forall i \leq 2m$, $\sum_{\tau \in \mathcal{J}} p_i(\tau) \leq z_i$
- (Max/min speed) $0 \leq \forall i \leq 2m$, if $l_i \neq l_{i+1}$, $(l_{i+1} - l_i)/v_{max} \leq z_i \leq (l_{i+1} - l_i)/v_{min}$. Remove the right inequality when $v_{min} = 0$.

4.5 Periodic Data Generation Case

So far we have discussed the non-periodic case, where each of the sensor nodes have fixed amount of data and the data mule travels across the sensor field only once. On the other hand, in applications that require long-term continuous monitoring, it may be more appropriate to assume that the data is generated periodically at a certain rate.

In this section we present algorithms for the constant speed and variable speed 1-D DMS problems under the periodic data generation case.

4.5.1 Problem Description

In the periodic data generation case, each sensor node generates data at a given rate and a data mule travels across the sensor field periodically. The data generation rate may differ among the sensor nodes. After each travel, the data mule needs to stop at the base station for a constant amount of time (called "stop time") to account for depositing the collected data and for refueling etc. The objective is to minimize the period, i.e., the time the data mule takes for each travel, since it largely affects the data delivery latency.

We introduce some notations for the periodic case. Let T_t denote the travel time of one period and T_b denote the stop time. Thus the length of one period is $T_t + T_b$. For the system to be stable, in each period of travel, the data mule needs to collect the data generated in one period. Otherwise data is accumulated at nodes, resulting in overflow of data.

4.5.2 Algorithms

Processor Demand Analysis

First we present an algorithm based on processor demand analysis. This algorithm applies to the constant speed model with simple location jobs, i.e., each location job has only one feasible location interval.

Let $e(\tau)$ denote the execution time of location job τ for one period. It is defined as follows:

$$e(\tau) \equiv \frac{\lambda(\tau)}{R}(T_t + T_b), \tag{4.1}$$

where $\lambda(\tau)$ is the data generation rate of the node represented by τ and R is the bandwidth, both of which are known constants.

Processor demand $g(I)$ for location interval I for one period is defined as $g(I) \equiv \sum_{\tau:I(\tau)\subseteq I} e(\tau)$, where $I(\tau)$ is the feasible location interval of location job τ. Let $g'(I)$ denote the processor demand for I for unit time, which is defined as follows:

$$g'(I) \equiv \frac{g(I)}{T_t + T_b} = \sum_{\tau:I(\tau)\subseteq I} \frac{\lambda(\tau)}{R}. \tag{4.2}$$

The set of location jobs is feasible if and only if the speed v of data mule satisfies

$$v \leq \min_{I\subseteq I_0} \frac{|I|}{g(I)} = \frac{1}{T_t + T_b} \min_{I\subseteq I_0} \frac{|I|}{g'(I)}, \tag{4.3}$$

where I_0 is the total travel interval.

When $T_b > 0$, we obtain the following constraint using $T_t = |I_0|/v$:

$$v \quad \leq \quad \left(\min_{I \subseteq I_0} \frac{|I|}{g'(I)} - |I_0| \right) \frac{1}{T_b}. \tag{4.4}$$

For a feasible solution to exist, the following must be satisfied:

$$|I_0| \quad < \quad \min_{I \subseteq I_0} \frac{|I|}{g'(I)}. \tag{4.5}$$

When this is satisfied, the maximum speed is the right hand side of (4.4). When this is not satisfied, it is not possible to collect data without loss.

When $T_b = 0$, we obtain the following from (4.3):

$$|I_0| \quad \leq \quad \min_{I \subseteq I_0} \frac{|I|}{g'(I)}. \tag{4.6}$$

Note that (4.6) contains neither v nor T_t. What it implies is, when this is satisfied, the speed of data mule can be arbitrary. This validates the experimental observation in [KSJ$^+$04] that the speed of data mule does not affect the throughput of data collection if the data mule travels the sensor field periodically without the stop time at the base station.

Online Algorithm

For the variable speed model with simple location jobs, we can use an online algorithm almost the same as EDD-WITH-STOP. A data mule moves at v_{max} while executing a job with the earliest deadline location. When a node still has data at its deadline, the data mule stops there and collects the data until the node becomes empty.

Linear Program Formulation

For the constant and variable speed models with general location jobs, we can use a linear program formulation.

The formulation is almost the same as the ones for the non-periodic cases with general location jobs (Section 4.3.2 for the constant speed model, Section 4.4.2 for the variable speed model), but the job completion constraint is replaced with the following one:

- (Job completion) $\forall \tau \in \mathcal{J}$, $\sum_{i=0}^{2m} p_i(\tau) = \left(\sum_{i=0}^{2m} z_i + T_b \right) \lambda(\tau)/R$, where R is the bandwidth of communication from each node to the data mule.

The right hand side is the amount of time to transmit the data generated in one period.

This linear program may be either infeasible or unbounded[3]. When it is infeasible, it is impossible to collect all data. When it is unbounded, the speed is arbitrary.

4.6 Connections with Speed Scaling Problems

In the 1-D DMS problem, we map each location point to a time point by determining the speed of the data mule and obtain a corresponding job scheduling problem. From another perspective, we can consider the 1-D DMS problem as a "scale-parameterized" scheduling problem, where some of the parameters are variables depending on some scaling factor. For the 1-D DMS case, the speed of data mule is the scaling factor and the location quantities such as release and deadline locations are the variables dependent on the scaling factor. As another example of scale-parameterized scheduling problems, we can think of the case where the release time and deadline are constant and the execution time is parameterized by a scaling factor. Interestingly enough, the resulting problem is analogous to processor speed scaling (as known as dynamic voltage scaling), which is a popular technique for reducing processor energy dissipation by lowering the supply voltage and operating frequency. Here we show the correspondence between the 1-D DMS problem and speed scaling problems.

4.6.1 Problem Definition of Speed Scaling Problems

A typical speed scaling problem is defined as follows, which is based on the definition in [ISG03]. The input is a tuple $(T, \{r_i, d_i, R_i\}, s_{min}, s_{max})$, where T is the end time, R_i is execution time in units of work, and s_{min}, s_{max} are the minimum and maximum processor speed. The most notable difference from normal scheduling problems is that the execution time is given as the number of units of work. This is because actual execution time will change depending on the processor speed $s(t)$, which represents the number of units of work the processor does per unit time. The output is a pair of functions $\mathcal{S} = (s(t), job(t))$, where $job(t)$ indicates which job is being executed at time t. A schedule is feasible if

$$\int_{r_j}^{d_j} s(t)\delta(job(t), j)dt \quad = \quad R_j \tag{4.7}$$

[3]It may be unbounded only in the constant speed model.

is satisfied for all j, where $\delta(x,y)$ is 1 if $x=y$ and 0 otherwise. The processor speed $s(t)$ always satisfies $s_{min} \leq s(t) \leq s_{max}$. We assume $T > d_j$ for all j.

The total energy cost $cost(\mathcal{S})$ is defined as

$$cost(\mathcal{S}) \;=\; \int_{t_0}^{t_1} P\left(s(t)\right) dt, \tag{4.8}$$

where $P(s)$ is the power consumption when the processor speed is s. $P(s)$ is a monotonically increasing convex function of s; typically $P(s) \propto s^2$. The objective is to find a feasible schedule $\mathcal{S}$ that minimizes $cost(\mathcal{S})$.

Normally speed scaling problems only deal with simple jobs that have one feasible interval per each job. Thus, once we determine $s(t)$, $job(t)$ is optimally determined by the EDD algorithm, as we have discussed in Section 4.2.1.

4.6.2 Constant Speed 1-D DMS and Static Speed Scaling

The Constant Speed 1-D DMS problem corresponds to static speed scaling (SSS) problem, where the processor speed $s(t)$ satisfies $s(t) = s_0$ for some s_0 and the problem is to minimize s_0.

We show how we can convert an instance of the Constant Speed 1-D DMS problem into the one of the SSS problem. For an instance of the Constant Speed 1-D DMS problem $\{L, \mathcal{J}\}$, where L is the total travel distance and $\mathcal{J}$ is a set of location jobs, given the data mule's speed v_0, we have an induced job scheduling problem in which a job τ is converted from a location job τ_L and has a release time $r(\tau_L)/v_0$, a deadline $d(\tau_L)/v_0$, and an execution time $e(\tau_L)$. The objective is to maximizing the speed v_0. Now, by expanding the time axis by a factor v_0, we can convert this induced job scheduling problem to an equivalent SSS problem $\{T, \mathcal{J}', [0, +\infty]\}$, where each job in $\mathcal{J}'$ has a release time $r(\tau_L)$, a deadline $d(\tau_L)$, and an execution time $e(\tau_L)$ in units of work. Note that the actual execution time is $v_0 e(\tau_L)$ in the time domain, which includes a variable v_0, but we can remove v_0 and make it a dimensionless parameter, since we see this problem as the SSS problem instead of a normal job scheduling problem. We can set T to a sufficiently large number. The objective is unchanged: to maximize v_0, or equivalently, to minimize $1/v_0$, while keeping $\mathcal{J}'$ feasible.

Conversely, we can convert an instance of the SSS problem into the one of the Constant Speed 1-D DMS problem in a similar way. The minimum and maximum processor speed in the SSS problem becomes a speed constraint on the 1-D DMS problem, and the problem is to find the maximum speed of the data mule within the speed range.

4.6.3 Variable Speed 1-D DMS and Dynamic Speed Scaling

A natural extension of the above argument is to compare the Variable Speed 1-D DMS problem with dynamic speed scaling (DSS) problem. However, the correspondence between these two are not as clear as the constant speed case.

An instance of the DSS problem is $\{T, \mathcal{J}_S, [s_{min}, s_{max}]\}$, which is same as the SSS problem. Based on the discussion in [YDS95], we can constrain the processor speed $s(t)$ to be piecewise constant with discontinuities only at the points in $P_r \cup P_d$, where P_r, P_d are sets of release time and deadline, respectively. Then we can represent $s(t)$ as $\{s_i, t_i\}$, meaning the processor runs at speed s_i for time duration t_i. Note t_i's are given constants.

The DSS problem is an optimization problem defined as follows. For each interval $[T_i, T_{i+1}]$ where $T_i = \sum_{k=0}^{i} t_k$, the variables are

- s_i: processor speed in this interval (in units of work per unit time)
- $a_i(\tau)$: units of work allocated to job τ in this interval

Then the problem is:

Minimize $\sum_i P(s_i)t_i$

Subject to

- (Minimum/maximum speed) $\forall i$, $s_{min} \le s_i \le s_{max}$
- (Feasible interval) $\forall \tau \in \mathcal{J}_S$, if $[T_i, T_{i+1}] \in \mathcal{I}(\tau)$, $a_i(\tau) = 0$
- (Job completion) $\forall \tau \in \mathcal{J}_S$, $\sum_i a_i(\tau) = e_c(\tau)$
- (Processor demand) $\forall i$, $\sum_{\tau \in \mathcal{J}_S} a_i(\tau) \le s_i t_i$

where $P(x)$ in the objective function is a function representing the power consumption for processor speed x. Thus the objective function is the total energy consumption.

Similarly as the constant speed case, s_i in the DSS problem corresponds to $1/v_i$ in the Variable Speed 1-D DMS problem, when we set the power function $P(s) = s$. It implies that the Variable Speed 1-D DMS problem for simple location jobs is easier than the DSS problem, since the former problem maps to a special case of the latter.

In dynamic speed scaling context, $P(s) = s^2$ is often assumed (for example in [IY98]), by assuming that the power is quadratically proportional to the supply voltage and that the frequency is linear to the supply voltage. Some papers ([ZBSF04], for example) use more precise model of the relation between frequency and supply voltage.

In fact, $P(s) = s$ leads to a trivial result in dynamic speed scaling context, since no matter how we change s, we can keep the total energy consumption unchanged. Especially when s can be 0, one simple optimal schedule is to use highest speed until a job finishes and go to sleep. However, the Variable Speed 1-D DMS problem is not as easy as this case, since $s = 0$ in speed scaling problem implies infinite speed of data mule, which is impossible. In addition, the case for general location jobs has not been studied in the context of dynamic speed scaling, as far as we know.

Using the correspondence between variable speed data mule scheduling and dynamic speed scaling, we can use Yao et al.'s optimal offline algorithm [YDS95], since it only assumes $P(s)$ to be a convex function of s. Indeed, our algorithm presented in Section 4.4.1 for simple location jobs is based on Yao et al.'s algorithm.

4.7 Related Work

We introduce related work on job scheduling and speed scaling, both of which are related to the 1-D DMS problem.

4.7.1 Job Scheduling

Our formulation of the 1-D DMS problem is most closely related to single processor preemptive scheduling with release times. For this problem, it is known that the Earliest Due Date (EDD) algorithm, in which a job with the earliest deadline among all available jobs is executed at any time slice, is optimal [Jac55, LL73, SSNB95]. However, we have also discussed non-standard cases in which each job may have multiple feasible intervals. As far as we know, there are only few studies on this case. Simons and Sipser [SS84] considered unit-length, non-preemptive jobs that have multiple feasible intervals, and showed the general problem is NP-complete. More recently, Shih et al. [SLC03] showed NP-hardness in case of preemptive jobs, but their assumption does not allow a job execution to continue over multiple feasible intervals: instead, partial work is lost at the end of each feasible interval if a job is incomplete. Chen et al. [CWSK05] present approximation algorithms for a similar problem in which the objective is to maximize the number of completed jobs both for preemptive and non-preemptive jobs, but they employ the non-continuation assumption above.

4.7.2 Speed Scaling

In their seminal paper [YDS95], Yao et al. presented a formal analysis of the speed scaling problem and designed an optimal offline algorithms and some online approximate algorithms. They employed a simple model, where they assumed that each job has a release time and deadline and that the processor speed is continuous. Ishihara and Yasuura [IY98] studied the cases with discrete processor speed. Pillai and Shin [PS01] discussed the speed scaling in the context of real time scheduling. In their model, the execution time is given as worst time execution time and also periodic tasks are assumed instead of jobs. They presented heuristic algorithms based on earliest deadline first (EDF) algorithm and rate monotonic (RM) algorithms [LL73]. Irani et al. [ISG03] employed a different processor model that has a sleep state. In their model, a processor consumes energy even when it is idle. By going into the sleep state, a processor can save the energy but there is a constant overhead for coming back to the active state. Irani et al. proposed some offline and online algorithms under these assumptions. Bansal et al. [BKP07] gave theoretical backgrounds on the optimality of Yao et al.'s algorithm [YDS95], and also proposed algorithms that optimizes the processor temperature.

4.8 Summary

In this chapter we discussed the basic cases of the 1-D DMS problem. The 1-D DMS problem consists of the speed control and job scheduling subproblems. We first extended the job scheduling problem for general jobs that have multiple feasible intervals and presented a linear program formulation. Then for the 1-D DMS problem with the constant speed and variable speed mobility models, we have presented optimal offline/online algorithms and also shown non-existence of online algorithms for some cases. We also analyzed the periodic data generation case. In the end we showed the correspondence between the 1-D DMS problem and speed scaling problems.

Acknowledgements This chapter, in part, has been submitted for publication as "Speed Control and Scheduling of Data Mules in Sensor Networks" by Ryo Sugihara and Rajesh K. Gupta in ACM Transactions on Sensor Networks [SG09e]. The dissertation author was the primary investigator and author of this paper.

This chapter, in part, has been submitted for publication as "Complexity of Motion Planning of Data Mule for Data Collection in Wireless Sensor Networks" by

Ryo Sugihara and Rajesh K. Gupta in Theoretical Computer Science [SG09a]. The dissertation author was the primary investigator and author of this paper.

Chapter 5

Motion Planning on Fixed Path: General Case

In this chapter we discuss the general case of the 1-D DMS problem. By the general case, we mean the Generalized mobility model (defined in Section 4.1.2), in which the data mule can change the speed under an acceleration constraint. We first define the Generalized 1-D DMS problem and show NP-hardness for general location jobs case, while the complexity for simple location jobs case remains open. We then present an approximation scheme for the case of simple location jobs. We also design a heuristic algorithm that always finds a feasible solution for any case. To evaluate the performance of the heuristic solutions, we analyze the lower bounds and compare them in simulation experiments. Finally, we discuss the connections with speed scaling problems and show how we can apply the approximation scheme to a special case of the speed scaling problem.

5.1 Generalized 1-D DMS Problem

The general case of the 1-D DMS problem, which we call the Generalized 1-D DMS problem, is defined by extending the definition of the original 1-D DMS problem presented in Section 4.1.3. An instance of the Generalized 1-D DMS problem is $(L, \mathcal{J}, v_{max}, a_{max})$, where

- $[0, L]$: total travel interval of the data mule on the location axis,
- $\mathcal{J}$: set of location jobs; i-th location job is characterized by

- $\mathcal{I}_i$: set of feasible location intervals, i.e., the location intervals in which the job is executable. Each feasible location interval $I \in \mathcal{I}_i$ is defined as a pair of release location $r(I)$ and deadline location $d(I)$,
- e_i: execution time.

- v_{max}: maximum speed constraint
- a_{max}: maximum absolute acceleration constraint

A solution to the problem is a time-speed profile $v(t)$ and a job schedule. Let T denote the total travel time. Then, the constraints on the motion of data mule are $0 \leq v(t) \leq v_{max}$, $|dv(t)/dt| \leq a_{max}$, $v(0) = v(T) = 0$, $\int_0^T v(t) = L$. In the optimization version of the Generalized 1-D DMS problem, the objective is to find a feasible solution that minimizes T. We can also consider a decision problem by adding a constant T to the input and asking whether there exists a time-speed profile such that the travel time is T and the induced job scheduling problem is feasible.

5.2 NP-Hardness for General Location Jobs Case

It is easy to see that there is no optimal online scheduling algorithm for Generalized 1-D DMS by considering a location job having a zero-length feasible location interval. As for offline scheduling, we have the following theorems about the hardness of the problem for general location jobs[1]:

Theorem 5.2.1. *Generalized 1-D DMS for general location jobs with k feasible location intervals is NP-hard for any fixed $k \geq 2$.*

Theorem 5.2.2. *Generalized 1-D DMS for general location jobs with k feasible location intervals is NP-hard in the strong sense for arbitrary k.*

The proofs are by reductions from PARTITION and 3-PARTITION, respectively. The basic ideas for constructing an instance of Generalized 1-D DMS from the one of (3-)PARTITION are as follows:

- Map each binary choice in the original problems to a "stop/skip" choice in the DMS problem.

[1]Complexity for simple location jobs case is still open.

  - Use a general location job with two zero-length feasible location intervals at separate locations: the data mule needs to stop at one of these locations.
  - Note that the data mule can also stop at both locations, but this is eliminated by the next idea.

- Set the total travel time to an appropriate minimum value.
  - If the data mule stops more often, it takes more time due to additional acceleration and deceleration.
  - We choose the value such that it is achievable only when the data mule stops the smallest possible times. This enforces the binary choices described above.

5.2.1 Proof of Theorem 5.2.1

For the decision version of Generalized 1-D DMS for general location jobs, we show a reduction from PARTITION for $k = 2$ case. It is easily extended for $k > 2$ cases as well. One way is to add to each location job a sufficient number of "padding intervals": zero-length feasible location intervals, all of which are located at the deadline location of the final feasible location interval of the job.

Let $A = \{a_1, ...a_n\}$ be the set of variables and $s(a_i) \in \mathbf{Z}^+$ be the size for each $a_i \in A$ in an arbitrary instance of PARTITION. Without loss of generality, we assume $s(a_i) = a_i$ in the rest of the proof.

Figure 5.1 shows an instance of Generalized 1-D DMS problem we construct. The set $\mathcal{J}$ of location jobs is represented as the union of three disjoint sets $\mathcal{J}_{L,1}, \mathcal{J}_{L,2}, \mathcal{J}_{L,3}$. Using the notation of each location job as "{(set of feasible location intervals), (execution time)}", these sets are defined as follows:

- Type-I: Subset choices:

$$\mathcal{J}_{L,1} = \bigcup_{i=1}^{n} \Big\{ \{[L_{i-1} + p_i], [L_n + L_{i-1} + p_i]\}, 1 \Big\},$$

- Type-II: Stop points:

$$\mathcal{J}_{L,2} = \bigcup_{i=0}^{2n} \Big\{ \{[L_i]\}, 1 \Big\},$$

- Type-III: Equalizers:

$$\mathcal{J}_{L,3} = \{\{[0, L_n]\}, 3S\} \cup \{\{[L_n, 2L_n]\}, 3S\},$$

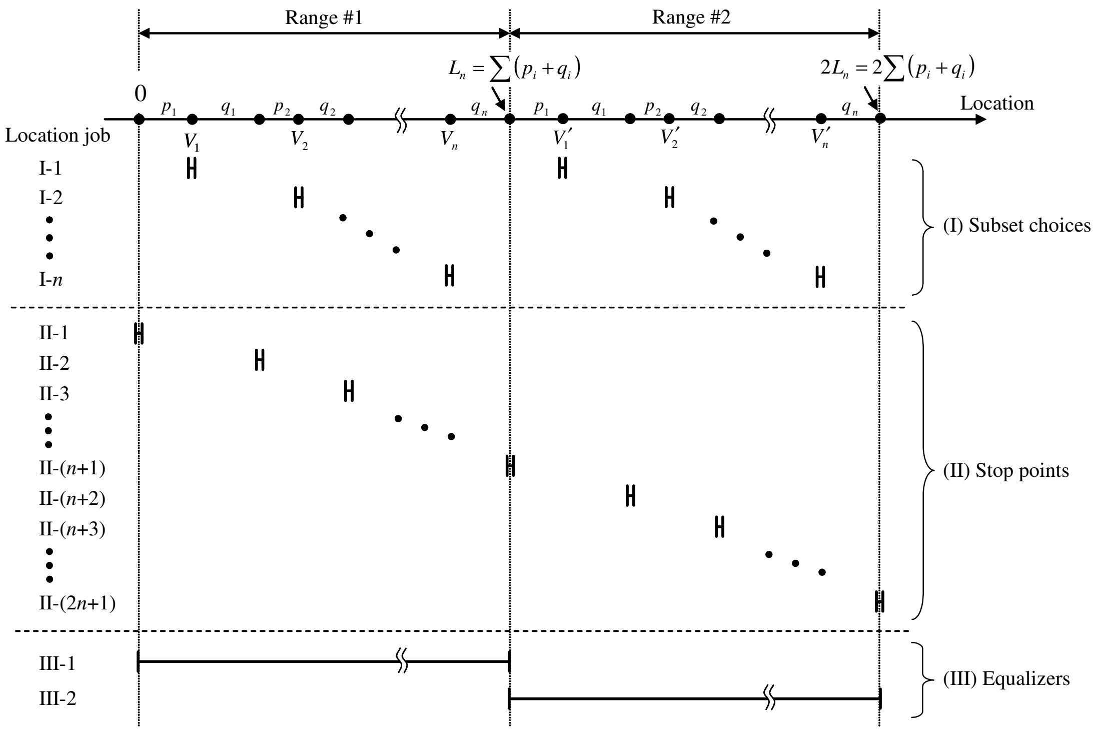


Figure 5.1: Reduction from PARTITION: Each row represents one location job. For type-I and type-II jobs, all feasible location intervals are zero-length.

where "$[x]$" is a short notation of a zero-length location interval $[x, x]$ and

$$L_i = \sum_{k=1}^{i}(p_k + q_k), \quad S = \sum_{i=1}^{n} a_i, \quad p_i = \frac{9}{16}a_i^2, \quad q_i = a_i^2.$$

We set the maximum speed v_{max} to a sufficiently large number, the maximum absolute acceleration $a_{max} = 1$, and the total travel interval $[0, L] = [0, 2L_n]$. For each location job to have two feasible location intervals, type-II and III jobs need one padding interval for each, but it is omitted from the above definition for clarity.

Intuitively, each of the type-I jobs corresponds to one variable in PARTITION. A type-I job has two zero-length feasible location intervals at points V_i in Range #1 and V_i' in Range #2 in the figure. Since both intervals are zero-length, the data mule needs to stop at either V_i or V_i' (or both) to execute the job. The central idea of this reduction is: at which of V_i or V_i' the data mule stops (and executes the job) corresponds to which of the subsets A' or $A - A'$ contains the variable a_i in PARTITION[2]. Type-II jobs restrict

[2]Later we will eliminate the possibility that the data mule stops at both V_i and V_i'.

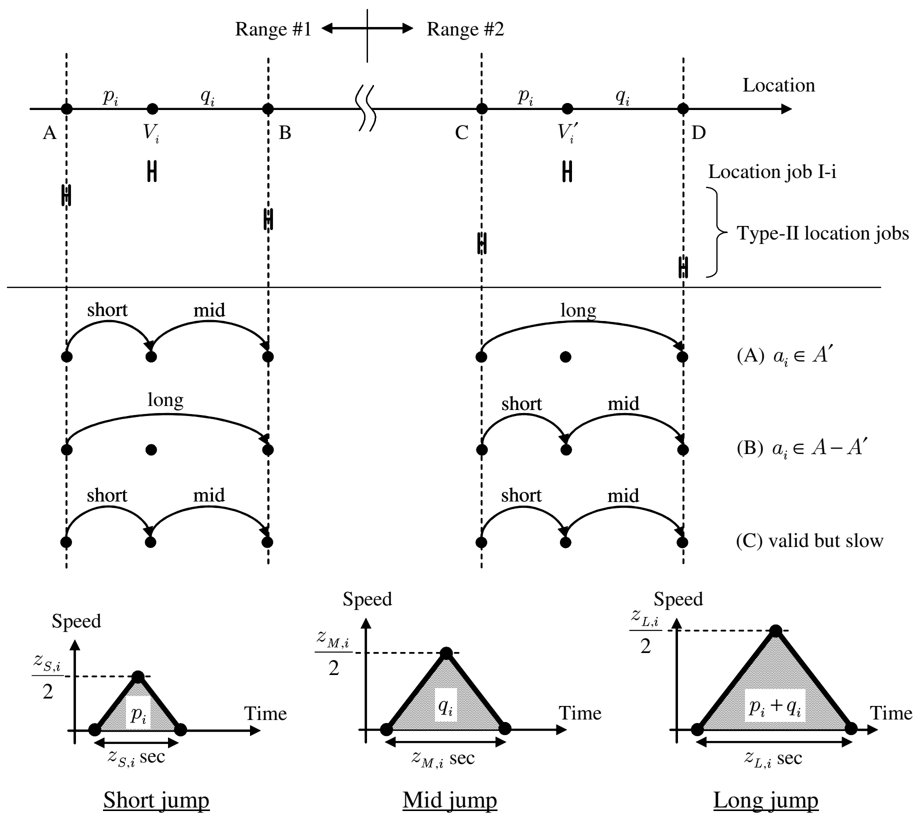


Figure 5.2: Possible travels and corresponding choices for i-th variable of PARTITION. The values of p_i, q_i are determined such that one short jump plus one mid jump ($z_{S,i} + z_{M,i}$) take longer than one long jump ($z_{L,i}$) exactly by a_i seconds, excluding the time the data mule is stopping to execute jobs.

possible movements of the data mule by forcing it to stop at certain locations. Since a type-II job has only one zero-length feasible location interval, the data mule needs to stop at the location to execute the job. We call these locations "stop points" afterwards. Type-III jobs work as equalizers: they force the data mule to spend equal time in Range #1 and Range #2.

Figure 5.2 is an excerpt related to a decision on one variable. It shows three possible travels to finish all the type-I and type-II jobs. The data mule must stop at stop points A, B, C, and D. To execute location job I-i, the data mule needs to stop either at V_i (Case A) or V_i' (Case B), or both (Case C). To minimize the travel time between two stop points, say, points A and B in Figure 5.2, the data mule should accelerate at the maximum acceleration ($a_{max} = 1$) first, then at the middle of A and B, decelerate at the negative maximum acceleration ($-a_{max}$). Let's call this movement a "jump." In

Figure 5.2, there are three possible distances to make such jumps: p_i, q_i, and $(p_i + q_i)$, and we denote the time it takes to make these jumps by $z_{S,i}$, $z_{M,i}$, and $z_{L,i}$, respectively. Then we have $p_i = z_{S,i}^2/4,\ q_i = z_{M,i}^2/4,\ p_i + q_i = z_{L,i}^2/4$, and thus

$$z_{S,i}^2 + z_{M,i}^2 \quad = \quad z_{L,i}^2 \tag{5.1}$$

For Cases (A) and (B) in Figure 5.2, the travel for the ranges AB and CD consists of one short jump, one mid jump, and one long jump, and thus it takes $(z_{S,i} + z_{M,i} + z_{L,i})$ in total, excluding the time to execute the jobs. For Case (C), it uses two short jumps and two mid jumps, and thus takes $2(z_{S,i} + z_{M,i})$, which is longer than the time in Cases (A) and (B). We choose p_i and q_i such that the travel time with one short jump plus one mid jump is slower exactly by a_i than the time with one long jump, where a_i is a variable in PARTITION. Then we have

$$z_{S,i} + z_{M,i} \quad = \quad z_{L,i} + a_i \tag{5.2}$$

One of the solutions satisfying both (5.1) and (5.2) is $(z_{S,i}, z_{M,i}, z_{L,i}) = (\frac{3}{2}a_i, 2a_i, \frac{5}{2}a_i)$. We use these parameters and obtain $p_i = \frac{9}{16}a_i^2,\ q_i = a_i^2$.

Now we consider the type-III jobs. Each type-III job has a feasible location interval covering the whole range. For a little while, we focus the discussion on the time the data mule is in motion and ignore the time it stops and executes the jobs. The fastest travel for each of two ranges takes $\sum_i z_{L,i} (= \frac{5}{2}S)$, when the data mule only uses long jumps. However, as discussed above, the data mule needs to stop either at V_i or V_i' for each i, and it additionally takes at least $\sum_i a_i (= S)$ between the start and the destination. Thus, excluding the time to stop and execute type-I and II jobs, the data mule takes at least $2\sum_i z_{L,i} + S (= 6S)$ to move from the start to the destination. Each of type-III jobs has the execution time equal to the half of this, and thereby we make the data mule spends the same time in each range, excluding the time the data mule is not moving.

Let T_M, T_S denote the total time the data mule is moving and stopping, respectively. As we discussed above, $T_M \geq 6S$. For T_S, we need to consider type-I and type-II jobs only, and get $T_S \geq n + 2n + 1 = 3n + 1$. To enforce the fastest possible travel, we set $T_S = 3n + 1$ and $T_M = 6S$. It eliminates the possibility of case (C) in Figure 5.2.

Finally, since the total travel time $T = T_S + T_M$, we set $T = 3n + 1 + 6S$, which is the minimum possible value. The construction above is done in time polynomial to the size of the original PARTITION problem.

($\Rightarrow$) For correctness, we first construct a feasible speed control plan and a feasible schedule, from a satisfying partition A' for the set A. A feasible speed control is represented by a collection of acceleration changing points S. An acceleration changing point P is represented as a tuple $\{x(P), z(P), v(P), a(P)\}$, where $x(P)$ is the current location, $z(P)$ is the time duration of keeping the same acceleration, $v(P)$ is the current speed, and $a(P)$ is the acceleration. We prepare a collection S_S of acceleration changing points for stopping to execute type-I and type-II jobs as follows. Note that the speed and acceleration are all zero for these.

$$\begin{aligned}
S_{1+} &= \bigcup_{\substack{i:1\le i\le n\\ a_i\in A'}} \{L_{i-1}+p_i, 1, 0, 0\},\\
S_{1-} &= \bigcup_{\substack{i:1\le i\le n\\ a_i\in A-A'}} \{L_n + L_{i-1}+p_i, 1, 0, 0\},\\
S_2 &= \bigcup_{i=1}^{2n+1} \{L_{i-1}, 1, 0, 0\},\\
S_S &= S_{1+}\cup S_{1-}\cup S_2.
\end{aligned}$$

$S_{1+}\cup S_{1-}$ corresponds to the set of all type-I jobs and S_2 corresponds to the set of all type-II jobs.

Next we construct a collection S_M of acceleration changing points for the movements as follows:

$$S_{M+} = \bigcup_{\substack{i:1\le i\le n\\ a_i\in A'}} \left\{ \begin{array}{l}
\underbrace{\{L_{i-1}, \frac{z_{S,i}}{2}, 0, 1\} \cup \{L_{i-1}+\frac{p_i}{2}, \frac{z_{S,i}}{2}, \frac{z_{S,i}}{2}, -1\}}_{\text{Short jump (Range \#1)}} \cup \\
\underbrace{\{L_{i-1}+p_i, \frac{z_{M,i}}{2}, 0, 1\} \cup \{L_{i-1}+p_i+\frac{q_i}{2}, \frac{z_{M,i}}{2}, \frac{z_{M,i}}{2}, -1\}}_{\text{Mid jump (Range \#1)}} \cup \\
\underbrace{\{L_n + L_{i-1}, \frac{z_{L,i}}{2}, 0, 1\} \cup \{L_n + L_{i-1}+\frac{p_i+q_i}{2}, \frac{z_{L,i}}{2}, \frac{z_{L,i}}{2}, -1\}}_{\text{Long jump (Range \#2)}}
\end{array} \right\},$$

$$S_{M-} = \bigcup_{\substack{i:1\le i\le n\\ a_i\in A-A'}} \left\{ \begin{array}{l}
\underbrace{\{L_{i-1}, \frac{z_{L,i}}{2}, 0, 1\} \cup \{L_{i-1}+\frac{p_i+q_i}{2}, \frac{z_{L,i}}{2}, \frac{z_{L,i}}{2}, -1\}}_{\text{Long jump (Range \#1)}} \cup \\
\underbrace{\{L_n + L_{i-1}, \frac{z_{S,i}}{2}, 0, 1\} \cup \{L_n + L_{i-1}+\frac{p_i}{2}, \frac{z_{S,i}}{2}, \frac{z_{S,i}}{2}, -1\}}_{\text{Short jump (Range \#2)}} \cup \\
\underbrace{\{L_n + L_{i-1}+p_i, \frac{z_{M,i}}{2}, 0, 1\} \cup \{L_n + L_{i-1}+p_i+\frac{q_i}{2}, \frac{z_{M,i}}{2}, \frac{z_{M,i}}{2}, -1\}}_{\text{Mid jump (Range \#2)}}
\end{array} \right\},$$

$$S_M = S_{M+}\cup S_{M-},$$

where S_{M+} and S_{M-} correspond to Cases (A) and (B) in Figure 5.2, respectively.

Then we construct a collection $S = S_S \cup S_M$. S represents a valid movement, since S_M assures the data mule to stop at every location contained in S_S and to make a connected movement between these locations. It also satisfies the one-way movement constraint and the maximum acceleration constraint. Let $||S||$ denote the time it takes to finish all the schedules in the collection S. Then, since $S_{1+}, S_{1-}, S_2, S_{M+}, S_{M-}$ are exclusive to each other, $||S_{1+} \cup S_{1-}|| = n$, $||S_2|| = 2n+1$, $||S_S|| = ||S_{1+} \cup S_{1-}|| + ||S_2|| = 3n+1$, $||S_M|| = ||S_{M+} \cup S_{M-}|| = 6S$, and thus $||S|| = ||S_S|| + ||S_M|| = 3n+1+6S$, which equals to T. Finally, all jobs can be completed by this schedule. Specifically, execute type-II jobs at the stop points and type-I jobs at the points representing variables if the data mule stops there, and while moving, execute type-III jobs. Therefore, S is a feasible speed control plan for the instance of Generalized 1-D DMS characterized by the set $\mathcal{J}$ of location jobs, sufficiently large v_{max}, the maximum absolute acceleration $a_{max} = 1$, the total travel interval $[0, 2L_n]$, and the total travel time $T = 3n + 1 + 6S$.

($\Leftarrow$) Next, we construct a satisfying partition for A from a feasible speed control plan for the above instance of Generalized 1-D DMS. We can immediately construct one as follows:

$$a_i \quad \in \quad \begin{cases} \quad A' & \text{if the data mule stops at } V_i \\ A - A' & \text{if the data mule stops at } V_i' \end{cases}$$

From the discussion on Figure 5.2, the data mule stops and executes a type-I job at either V_i or V_i' $(1 \leq i \leq n)$ but not both, which makes the above construction valid. In addition, it stops at every stop point and executes a type-II job. From the conditions that both of the type-III jobs are completed, the movement of the data mule takes additional time[3] by $S/2$ compared to the fastest possible travel, which is realized by using long jumps only. Since the additional moving time incurred by stopping at each V_i or V_i' is equal to a_i, we obtain $\sum_{a_i \in A'} a_i = \sum_{a_i \in A - A'} a_i = S/2$, which means A' is a satisfying partition.

5.2.2 Proof of Theorem 5.2.2

We show a reduction from 3-PARTITION. Let $A = \{a_1, ..., a_{3m}\}$ be the set of variables, $B \in \mathbf{Z}^+$ be the bound, and $s(a_i) \in \mathbf{Z}^+$ be the size for each $a_i \in A$ such that $B/4 \leq s(a_i) \leq B/2$ and $\sum_{a_i \in A} s(a) = mB$ in an arbitrary instance of 3-PARTITION. For simplicity we assume $s(a_i) = a_i$.

[3]Note that it only accounts for the moving time, i.e., time to execute type-I jobs is excluded.

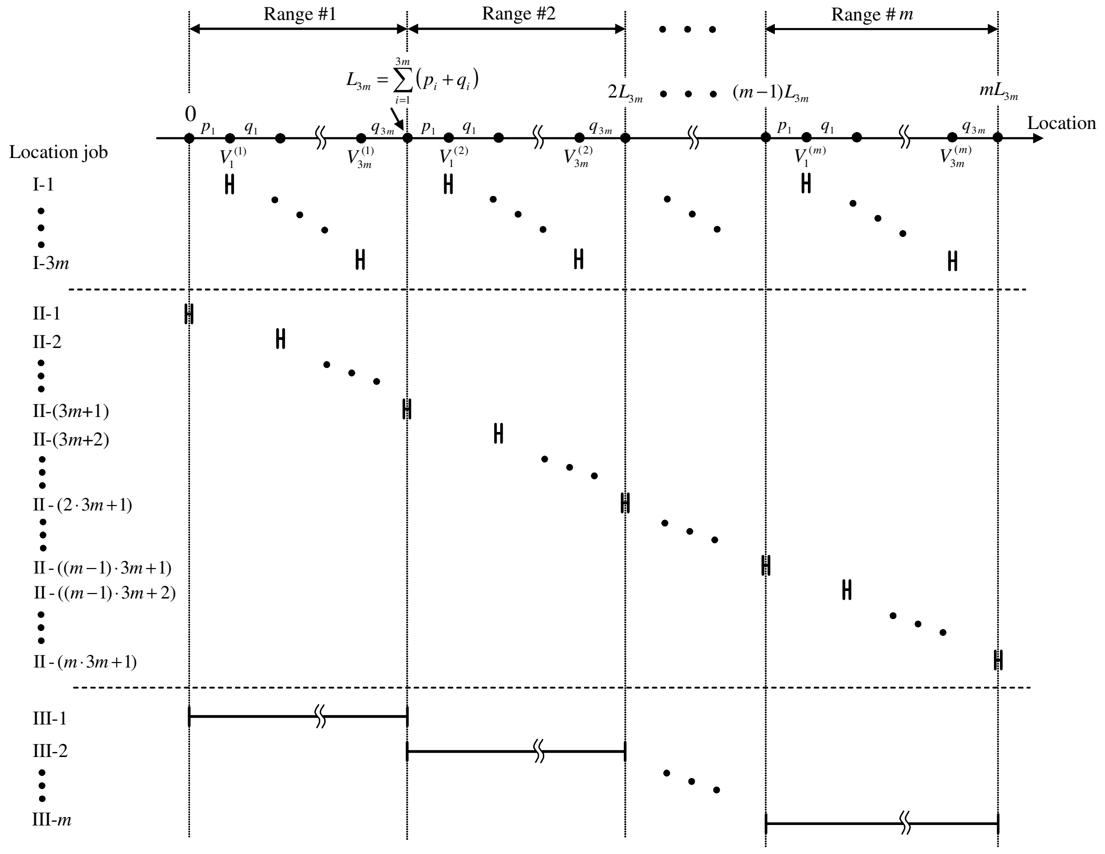


Figure 5.3: Reduction from 3-PARTITION: Each row represents one location job. For type-I and type-II jobs, all feasible location intervals are zero-length.

Figure 5.3 shows an instance of Generalized 1-D DMS we construct. The location axis is divided into m ranges, each of which is analogous to the one used in the proof for the case of fixed k. The set $\mathcal{J}$ of location jobs is the union of the following sets:

- Type-I: Subset choices:

$$\mathcal{J}_{L,1} = \bigcup_{i=1}^{3m} \left\{ \left\{ \bigcup_{k=0}^{m-1} [kL_{3m} + L_{i-1} + p_i] \right\}, 1 \right\},$$

- Type-II: Stop points:

$$\mathcal{J}_{L,2} = \bigcup_{k=0}^{m-1} \bigcup_{i=1}^{3m} \Big\{ \{[kL_{3m} + L_{i-1}]\}, 1 \Big\} \cup \{\{[mL_{3m}]\}, 1\},$$

- Type-III: Equalizers:

$$\mathcal{J}_{L,3} = \bigcup_{k=0}^{m-1} \left\{ \{[kL_{3m}, (k+1)L_{3m}]\}, \left(\frac{5}{2}m + 1\right) B \right\},$$

where

$$L_i = \sum_{k=1}^{i}(p_k + q_k), \quad B = \frac{1}{m}\sum_{i=1}^{3m} a_i, \quad p_i = \frac{9}{16}a_i^2, \quad q_i = a_i^2.$$

We set the maximum speed v_{max} to a sufficiently large number, the maximum absolute acceleration $a_{max} = 1$, and the total travel interval $[0, L] = [0, mL_{3m}]$. Choices of p_i and q_i are based on the same discussion as before. For the fastest travel, the data mule needs to stop at only one of $V_i^{(k)}$, and stopping at $V_i^{(k)}$ takes additional a_i seconds compared to skipping it, excluding the time to execute a type-I job.

The fastest travel for each range takes $\sum z_{L,i} = \frac{5}{2}mB$. For the whole range, it takes additional $\sum a_i = mB$ to execute all type-I jobs. Thus, the data mule spends $m{\cdot}\frac{5}{2}mB{+}mB = (\frac{5}{2}m^2{+}m)B$ to move from the start to the destination. For the total time the data mule stops (denoted T_S), since there are $3m$ type-I jobs and $3m^2{+}1$ type-II jobs, $T_S = 3m^2{+}3m{+}1$ seconds in total. Therefore we get $T = 3m^2{+}3m{+}1{+}(\frac{5}{2}m^2{+}m)B$. The construction above is done in time polynomial to the size of the original 3-PARTITION problem.

Correctness is proved in a similar way as the case of fixed k.

5.3 Approximation Scheme for Simple Location Jobs Case

In this section we present a polynomial time approximation scheme[4] for Generalized 1-D DMS for simple location jobs case. The complexity for this problem is still open, but is presumably hard since there is no simple non-trivial algorithm that always yields feasible solutions.

We design the approximation scheme by extending the one for kinodynamic motion planning [DXCR93] to a scheduling problem. The technique we used is a rather standard one: to discretize the solution space and use dynamic programming, but it requires an elaborate analysis to determine the step size so that the approximation ratio is guaranteed. In the rest of the section, we first give a summary of the result and comparisons to other techniques of designing a PTAS, and then discuss the details.

Following theorem is the main result of this section:

[4]Technically, our algorithm is not a PTAS: the running time is polynomial to the number of location jobs and approximation factor $(1/\epsilon)$ and is pseudopolynomial to the dynamics parameters. However, we just use the term PTAS for simplicity.

Theorem 5.3.1. *For an instance of Generalized 1-D DMS problem* $(L, \{r_i, d_i, e_i\}, v_{max}, a_{max})$*, there exists an algorithm that finds a trajectory* Γ_d *that satisfies* $|\Gamma_d| \leq (1 + 2\epsilon)|\Gamma_{OPT}|$ *for any positive constant* $\epsilon < 1$*, where* $|\Gamma|$ *denotes the travel time of* Γ*, and runs in time*

$$O\left(n^7 \left(\frac{1}{\epsilon}\right)^6 \left(\frac{\sum_i e_i}{\min_i\{e_i\}}\right)^2 \frac{v_{max}^2}{a_{max}L}\right), \tag{5.3}$$

where n *is the number of location jobs.*

The running time is polynomial to the number of jobs and $(1/\epsilon)$ and pseudopolynomial to dynamics parameters. It is also polynomial to the ratio of the total execution time to the minimum of execution time.

As mentioned earlier, our approximation scheme extends Donald et al.'s work [DXCR93] by incorporating feasibility constraints into the algorithm. The basic approach is to discretize the solution space and use dynamic program. Although this is a standard approach in designing a PTAS for several problems including scheduling, our method is characteristic in the following ways:

- Discretization is applied in a very different way from [DXCR93]. This is due to the fact that, in addition to the spatial constraints as in kinodynamic motion planning, the 1-D DMS problem also has temporal constraints expressed by feasibility condition. We apply discretization only to the speed and the execution time at certain location points, and not to the whole location axis and the travel time. We determine the quantization steps to guarantee that there exists a discrete trajectory that spends equal time to any continuous trajectory in each location interval, except for short location intervals.

- Short location intervals are handled separately so that the running time is bounded by polynomial time. The approach is different from the PTAS for bin packing [Vaz03], where small items are used for filling the space after packing large items, and the PTAS for real time scheduling with fixed priorities [ER08], where small tasks are grouped into a large task. In the 1-D DMS problem, we cannot move or group the short intervals, since location jobs are inherent in these intervals. Instead, in short intervals, we allow a discrete trajectory to spend more time than a continuous one, but also guarantee that the additional time is upper-bounded.

5.3.1 Preliminaries

Definitions

First we define *trajectory* to represent the motion of the data mule.

Definition 5.3.2. *A trajectory from (l_0, v_0) to (l_1, v_1) is a path in location-speed space starting at location l_0 with speed v_0 and ending at location l_1 with speed v_1. A trajectory is a-accel-bounded when the acceleration is in the range $[-a, a]$ at any time for a positive constant a. A trajectory is v-speed-bounded when the speed is in the range $[0, v]$ at any time for a positive constant v. A trajectory is (a, v)-bounded when it is a-accel-bounded and v-speed-bounded.*

Next we define equivalence of two trajectories. A trajectory is equivalent to another one when the induced job scheduling problems are identical. This can be defined by using the notion of dominant points.

Definition 5.3.3. *A dominant point is a location that is either an endpoint of the total travel interval (i.e., start or end location) or an endpoint of the feasible location interval of a location job (i.e., release or deadline location).*

Definition 5.3.4. *Two trajectories are equivalent if the arrival and departure time for all dominant points are identical in two trajectories.*

Finally, we define *feasible trajectory*. We also define *c-stretched feasible* trajectory to quantify the time margin to maintain the feasibility. The notion of stretched feasibility is similar in spirit to resource augmentation (e.g., [BCK$^+$07]).

Definition 5.3.5. *A trajectory is feasible for $\mathcal{J}$ when the induced job scheduling problem has a feasible schedule. A trajectory is c-stretched feasible for $\mathcal{J}$ if the trajectory is feasible for a modified set of location jobs $\mathcal{J}'$ in which each execution time is multiplied by a positive constant c.*

Feasibility Test by Cumulative Execution Time

We use *cumulative execution time* for feasibility testing. Specifically, for a given trajectory Γ, we define a cumulative execution time function $t_\Gamma(x)$ at location x. We first define a function $f_\Gamma(x)$ that maps location x to the departure time from x based on the speed change in Γ. Note that the arrival and departure time may be different when the speed at x is zero. Using $f_\Gamma(x)$, we can define $t_\Gamma(x)$ as the total executed portion of all jobs at time $f_\Gamma(x)$. In other words, $t_\Gamma(x)$ is the total non-idle time when the data mule departs from location x. By extending Definition 5.3.5, we call $t_\Gamma(x)$ is (c-stretched) feasible when it corresponds to a (c-stretched) feasible schedule for the induced job scheduling problem.

Although $t_\Gamma(x)$ is not unique for Γ, we can define the optimal $t^*_\Gamma(x)$ in which the allocation is based on the EDD algorithm (see Section 4.2.1). For such $t^*_\Gamma(x)$, $t_\Gamma(x) \leq t^*_\Gamma(x)$ is satisfied for any x. Then we can define the feasibility condition as follows:

Lemma 5.3.6. *A trajectory Γ is feasible for $\mathcal{J}$ iff the optimal cumulative execution time function satisfies $t^*_\Gamma(d_i) \geq \sum_{j:d_j \leq d_i} e_j$ for all deadline locations d_i.*

Proof. ($\Rightarrow$) When the trajectory Γ is feasible, we show $t^*_\Gamma(x)$ satisfies the condition. Since Γ is a feasible trajectory, when the data mule departs from d_i, all jobs that have deadline locations at or before d_i have been finished in the allocation based on the EDD algorithm. The sum of execution time of such jobs is $\sum_{j:d_j \leq d_i} e_j$. As other jobs can be executed in the EDD algorithm as well, $t^*_\Gamma(d_i)$ is equal to or greater than this.

($\Leftarrow$) When the condition is satisfied, we show that the trajectory is feasible. Until d_i, at least time $\sum_{j:d_j \leq d_i} e_j$ has been allocated to jobs. Since the EDD algorithm always allocates time to the job with earliest deadline, all of the jobs that have deadline locations at or before d_i have been finished. Since this holds for all deadline locations, deadline constraint is never violated and thus the trajectory is feasible. □

Trajectories between Two Points

Let $L_{min}(a, v, z, v_0, v_1)$ and $L_{max}(a, v, z, v_0, v_1)$ denote the minimum and maximum travel distance of an (a, v)-bounded trajectory from $(0, v_0)$ that ends after time z with speed v_1. These are expressed in a closed form as follows:

$$L_{min}(a, v, z, v_0, v_1) = \begin{cases} \dfrac{1}{4a}\left(-a^2z^2 + 2a(v_0+v_1)z + (v_1-v_0)^2\right) & \text{If } v_0 + v_1 \geq az \\ \dfrac{v_0^2+v_1^2}{2a} & \text{Otherwise} \end{cases}$$

$$L_{max}(a, v, z, v_0, v_1) = \begin{cases} \dfrac{1}{4a}\left(a^2z^2 + 2a(v_0+v_1)z - (v_1-v_0)^2\right) & \text{If } v_0 + v_1 \leq 2v - az \\ vz - \dfrac{v^2}{a} + \dfrac{(v_0+v_1)v}{a} - \dfrac{v_0^2+v_1^2}{2a} & \text{Otherwise} \end{cases}$$

We show the derivations of these equations later. The following lemma states that we can construct any trajectory in between these two:

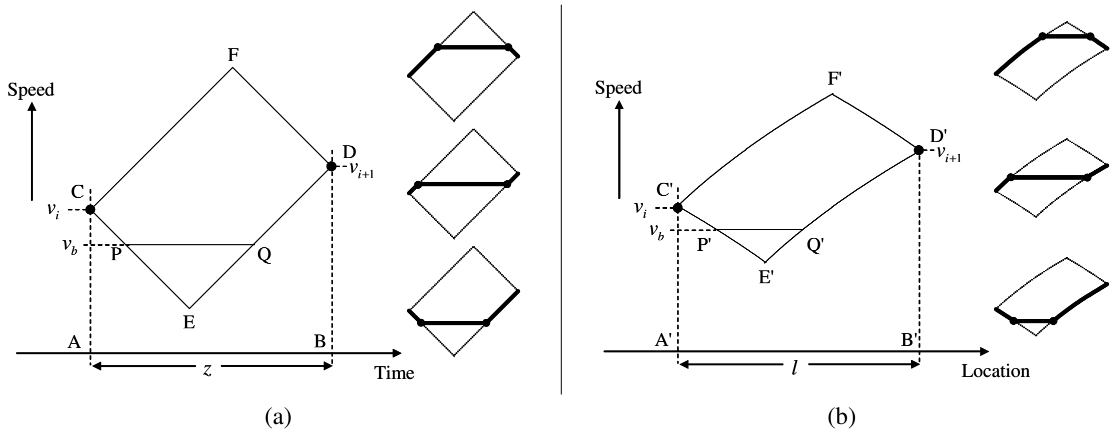


Figure 5.4: Idea for constructing trajectories with (a) arbitrary travel distance (Lemma 5.3.7) and (b) arbitrary travel time (Lemma 5.3.8). The small figures show the shapes of trajectory for different v_b's.

Lemma 5.3.7. *For any l satisfying $L_{min}(a, v, z, v_0, v_1) \leq l \leq L_{max}(a, v, z, v_0, v_1)$, there exists an (a, v)-bounded trajectory Γ from $(v_0, 0)$ to (v_1, l) such that $|\Gamma| = z$.*

Proof. We fix the travel time to z and find a trajectory whose travel distance is l. Note that the area under a trajectory in time-speed graph is the travel distance. We consider the trajectories with 3-phase acceleration changes, as shown in the right side of Figure 5.4(a). In the time-speed graph shown in Figure 5.4(a), the slope of CF and ED is $+a$ and that of CE and FD is $-a$. Then the area of ACEDB is $L_{min}(a, v, z, v_0, v_1)$ and that of ACFDB is $L_{max}(a, v, z, v_0, v_1)$. Now we consider a horizontal line segment PQ across the rectangle at $v = v_b$ and suppose that a trajectory first starts from C and follows the line segment CP (i.e., acceleration at $-a$), then follows PQ (i.e., no acceleration), and follows QD (i.e., acceleration at $+a$) to reach D. In this trajectory, the travel distance is the area ACPQDB. When we change v_b to $v_b + \delta$, the travel distance also increases, since the area ACPQDB also increases while changing the shape as shown in the right side of Figure 5.4(a). Since we can change v_b continuously, there exists v_b^* such that the travel distance equals l, and we can use that as Γ. We only show the case in which the speed at E is nonnegative and that at F is at most v, but the same construction idea is used for other cases as well. $\square$

Similarly, we consider the trajectories with the minimum and maximum travel time. Let $T_{min}(a, v, l, v_0, v_1)$ and $T_{max}(a, v, l, v_0, v_1)$ denote the minimum and maximum travel time of (a, v)-bounded trajectory from $(0, v_0)$ to (l, v_1). These are expressed in a closed form as follows:

$$T_{min}(a, v, l, v_0, v_1) = \begin{cases} \dfrac{2}{a}\sqrt{al + \dfrac{v_0^2 + v_1^2}{2}} - \dfrac{v_0 + v_1}{a} & \text{If } v_0^2 + v_1^2 < 2v^2 - 2al \\ \dfrac{l}{v} + \dfrac{v - v_0 - v_1}{a} + \dfrac{v_0^2 + v_1^2}{2av} & \text{Otherwise} \end{cases}$$

$$T_{max}(a, v, l, v_0, v_1) = \begin{cases} -\dfrac{2}{a}\sqrt{-al + \dfrac{v_0^2 + v_1^2}{2}} + \dfrac{v_0 + v_1}{a} & \text{If } v_0^2 + v_1^2 > 2al \\ +\infty & \text{Otherwise} \end{cases}$$

The derivations of these equations are shown later. We have a lemma similar to Lemma 5.3.7:

Lemma 5.3.8. *For any z satisfying $T_{min}(a, v, l, v_0, v_1) \le z \le T_{max}(a, v, l, v_0, v_1)$, there exists an (a, v)-bounded trajectory Γ from $(v_0, 0)$ to (v_1, l) such that $|\Gamma| = z$.*

Proof. We use the same idea as the proof of Lemma 5.3.7, but in the location-speed space as shown in Figure 5.4(b). In the figure, the curves C'F' and E'D' represent the acceleration at $+a$ and C'E' and F'D' represent the acceleration at $-a$. We consider a horizontal line segment P'Q' at $v = v_b$ and a trajectory C'P'Q'D' that follows the perimeter of the region and the line. Note that in this case, the area under a trajectory does not mean anything. However, when we increase v_b, the travel time is decreased, since the speed of the new trajectory is faster than or equal to that of the original trajectory at any location. Since we can change v_b continuously, there exists v_b^* such that the travel time equals z. $\square$

Derivations of L_{min}, L_{max}, T_{min}, T_{max} We first show the derivations of L_{min} and L_{max}, which are the maximum and minimum travel distance when the starting/ending speed and travel time are given. In Figure 5.5, set $z_a = z_b = z_c = z_d = z$.

When the maximum acceleration is constrained to a, the maximum travel distance is achieved by first accelerating at the maximum acceleration a and then decelerating at a. This is Case (a) in Figure 5.5. Let v_m denote the maximum speed in this travel. Since v_m satisfies $(v_m - v_0)/a + (v_1 - v_m)/(-a) = z$,

$$v_m = \frac{az}{2} + \frac{v_0 + v_1}{2}. \tag{5.4}$$

When $v_m \le v$, i.e., $v_0 + v_1 \le 2v - az$, we have $v_0 + az_{a1} = v_m$ and thus

$$z_{a1} = \frac{z}{2} + \frac{v_1 - v_0}{2a}. \tag{5.5}$$

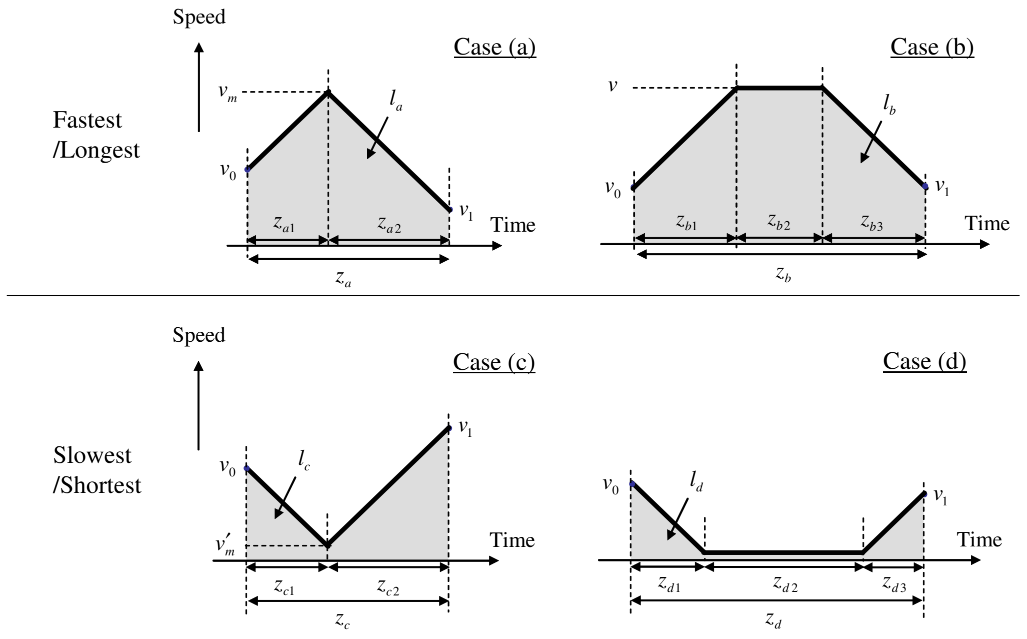


Figure 5.5: Time-speed profiles for the trajectories from $(l, v) = (0, v_0)$ to (l, v_1).

Therefore the maximum travel distance is

$$\begin{aligned} l_a &= \frac{v_0 + v_m}{2} z_{a1} + \frac{v_1 + v_m}{2} z_{a2} \\ &= \frac{1}{4a}\left(a^2 z^2 + 2a(v_0 + v_1)z - (v_1 - v_0)^2\right). \end{aligned} \tag{5.6}$$

When $v_m > v$, i.e., $v_0 + v_1 > 2v - az$, the time-speed profile is like Case (b) in the figure. In this case, since $v_0 + az_{b1} = v_1 + az_{b3} = v$, we have $z_{b1} = (v - v_0)/a$ and $z_{b3} = (v - v_1)/a$. Then the maximum travel distance is

$$\begin{aligned} l_b &= \frac{v_0 + v}{2} z_{b1} + vz_{b2} + \frac{v_1 + v}{2} z_{b3} \\ &= vt - \frac{v^2}{a} + \frac{(v_0 + v_1)v}{a} - \frac{v_0^2 + v_1^2}{2a}. \end{aligned} \tag{5.7}$$

The shortest travel distance for a given travel time is achieved by first decelerating at a and then accelerating at a. This is Case (c) in the figure. Let v_m' denote the minimum speed in this travel. Since v_m' satisfies $(v_0 - v_m)/a + (v_1 - v_m)/a = z$,

$$v_m' = \frac{v_0 + v_1}{2} - \frac{az}{2}. \tag{5.8}$$

When $v_m' \geq 0$, i.e., $v_0 + v_1 \geq az$, we have $v_0 - az_{c1} = v_m'$ and thus

$$z_{c1} = \frac{z}{2} + \frac{v_0 - v_1}{2a}. \tag{5.9}$$

Then the minimum travel distance is

$$\begin{aligned} l_c &= \frac{v_0 + v'_m}{2} z_{c1} + \frac{v_1 + v'_m}{2} z_{c2} \\ &= \frac{1}{4a}\left(-a^2 z^2 + 2a(v_0 + v_1)z + (v_1 - v_0)^2\right). \end{aligned} \tag{5.10}$$

Finally, when $v'_m < 0$, i.e., $v_0 + v_1 < az$, the time-speed profile is like Case (d) with the lowest speed (during z_{d2}) is actually zero. In this case, since $z_{d1} = v_0/a$ and $z_{d3} = v_1/a$, the minimum travel distance is

$$\begin{aligned} l_d &= \frac{v_0}{2} z_{d1} + \frac{v_1}{2} z_{d3} \\ &= \frac{v_0^2 + v_1^2}{2a}. \end{aligned} \tag{5.11}$$

Next we show the derivations of T_{min} and T_{max}, which are the maximum and minimum travel time when the starting/ending speed and the travel distance are given. In Figure 5.5, set $l_a = l_b = l_c = l_d = l$.

The analysis is very similar to the previous one. For Case (a) in Figure 5.5, since the maximum speed v_m satisfies $(v_m^2 - v_0^2) + (v_m^2 - v_1^2) = 2al$,

$$v_m = \sqrt{al + \frac{v_0^2 + v_1^2}{2}}. \tag{5.12}$$

When $v_m < v$, i.e., $v_0^2 + v_1^2 < 2v^2 - 2al$, the minimum travel time is

$$\begin{aligned} z_a &= \frac{v_m - v_0}{a} + \frac{v_1 - v_m}{-a} \\ &= \frac{2}{a}\sqrt{al + \frac{v_0^2 + v_1^2}{2}} - \frac{v_0 + v_1}{a}. \end{aligned} \tag{5.13}$$

When $v_m \geq v$, i.e., $v_0^2 + v_1^2 \geq 2v^2 - 2al$, the time-speed profile is like Case (b) in the figure. In this case, since $z_{b1} = (v - v_0)/a$ and $z_{b3} = (v - v_1)/a$,

$$\begin{aligned} l &= \frac{v_0 + v}{2} z_{b1} + v z_{b2} + \frac{v_1 + v}{2} z_{b3} \\ &= \frac{1}{2a}(2v^2 - v_0^2 - v_1^2) + v z_{b2}. \end{aligned} \tag{5.14}$$

Then we have

$$z_{b2} = \frac{l}{v} - \frac{v}{a} - \frac{v_0^2 + v_1^2}{2av}. \tag{5.15}$$

Therefore the minimum travel time is

$$z_b = \frac{l}{v} + \frac{v - v_0 - v_1}{a} + \frac{v_0^2 + v_1^2}{2av}. \tag{5.16}$$

The slowest way to go from $(l, v) = (0, v_0)$ to (l, v_1) is to first decelerate at a and then to accelerate at a. This is Case (c) in the figure. Since v'_m satisfies $(v_0^2 - v'^2_m) + (v_1^2 - v'^2_m) = 2al$,

$$v'_m = \sqrt{-al + \frac{v_0^2 + v_1^2}{2}}. \tag{5.17}$$

When $-al + (v_0^2 + v_1^2)/2 > 0$, i.e., $v_0^2 + v_1^2 > 2al$, the maximum travel time is

$$z_c = -\frac{2}{a}\sqrt{-al + \frac{v_0^2 + v_1^2}{2}} + \frac{v_0 + v_1}{a} \tag{5.18}$$

When $v_0^2 + v_1^2 \leq 2al$, the time-speed profile is like Case (d). In this case, the speed during the constant speed movement can be arbitrarily small. Thus z_{d2} can be arbitrarily large and the maximum travel time z_d is unbounded.

5.3.2 Slower Trajectory

We first show that, for any trajectory, there exists another one that is slower and constrained by slightly tighter acceleration and speed bounds. Then we determine the size of speed quantization step for normal intervals such that there exists a discrete trajectory with the original bounds that is equivalent to this slower one with the tighter bounds. For short intervals, we use a different upper-bounding condition to determine the step size. Finally, we quantize the cumulative execution time and design a dynamic programming algorithm.

The following lemma gives a slower trajectory with tighter bounds:

Lemma 5.3.9. *Let ϵ be a small constant satisfying $0 < \epsilon < 1$. For any (a_{max}, v_{max})-bounded trajectory Γ, there exists an $(a_{max}/(1+\epsilon)^2, v_{max}/(1+\epsilon))$-bounded $(1+\epsilon)$-stretched feasible trajectory.*

Proof. Consider $(a_{max}/(1+\epsilon)^2, v_{max}/(1+\epsilon))$-bounded trajectory Γ_s with speed $v_s(t) = v(t/(1+\epsilon))/(1+\epsilon)$ and acceleration $a_s(t) = a(t/(1+\epsilon)^2)/(1+\epsilon)^2$, where $v(t), a(t)$ are the speed and acceleration of Γ at time t. Then, for any location intervals, Γ_s spends $(1+\epsilon)$ times more time than Γ does. Therefore Γ_s is $(1+\epsilon)$-stretched feasible and the lemma follows. $\square$

5.3.3 Speed Quantization

We first present the basic idea for determining the size of quantization step for normal intervals. Then, for short intervals, we present a different way.

Normal intervals

The following lemma states that, for any trajectory, by slightly relaxing the bounds on acceleration and speed, we can construct an equivalent trajectory with its speed on each dominant point being equal to or less than that of the original one and also being an integer multiple of some quantization step.

Lemma 5.3.10. *Let* ϵ *be a small constant satisfying* $0 < \epsilon < 1$*. Suppose, for an* (a_{max}, v_{max})*-bounded trajectory* Γ *from* $(0, v_0)$ *to* (l, v_1)*, an* $(a_{max}/(1+\epsilon)^2, v_{max}/(1+\epsilon))$*-bounded trajectory* Γ_s *from* $(0, v_0)$ *to* (l, v_1) *is constructed as stated in Lemma 5.3.9. Then, for such* Γ_s *and positive constants* q_0, q_1*, there exists an* (a_{max}, v_{max})*-bounded trajectory* Γ_q *from* $(0, v_0 - q_0)$ *to* $(l, v_1 - q_1)$ *that satisfies* $|\Gamma_q| = |\Gamma_s|$ *if*

$$\max\{q_0, q_1\} \quad \leq \quad \min\left\{\frac{\epsilon}{8}\frac{a_{max}l}{v_{max}},\ \frac{\epsilon^2 v_{max}}{4}\right\}. \tag{5.19}$$

Proof. We first show that the following two statements are equivalent:

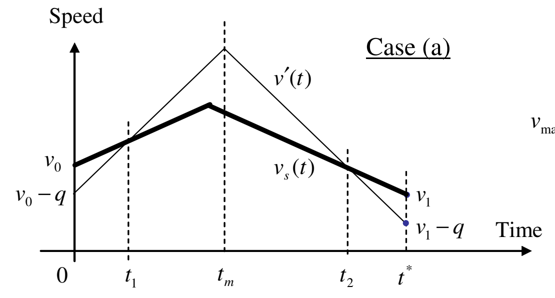
1. For any Γ_s, there exists Γ_q such that $|\Gamma_q| = |\Gamma_s|$.
2. $\min|\Gamma_q| \leq \min|\Gamma_s|$ and $\max|\Gamma_q| \geq \max|\Gamma_s|$.

From the first statement to the second one is trivial. The other direction is also true due to the fact that we can construct a trajectory having any travel time between $\min|\Gamma_q|$ and $\max|\Gamma_q|$ (Lemma 5.3.8).

We also claim that $\max|\Gamma_q| \geq \max|\Gamma_s|$ is satisfied for any positive q_0, q_1. This is because we can construct Γ_q whose speed is smaller than or equal to that of Γ_s at any location, and also because $q_0, q_1 > 0$. Thus, to prove the lemma, we need to show $\min|\Gamma_q| \leq \min|\Gamma_s|$.

Now we consider another (a_{max}, v_{max})-bounded trajectory Γ' from $(0, v_0 - q_0)$ to $(l', v_1 - q_1)$ and $|\Gamma'| = t^*$, where l' is a positive variable and $t^* = \min|\Gamma_s|$. The difference between Γ' and Γ is that the travel distance of Γ' (i.e., l') may not be equal to l. Then, $\min|\Gamma_q| \leq \min|\Gamma_s| \Leftrightarrow \max\{l'\} \geq l$, which is shown from Lemma 5.3.7. To summarize, we want to find a condition on q so that $\max\{l'\} \geq l$ is always satisfied, where $q = \max\{q_0, q_1\}$.

The technique used in the following argument is similar to [DXCR93]. However, a major difference is that we do not discretize the time at this point. To find the conditions to satisfy $\max\{l'\} \geq l$, it is sufficient to consider Γ_s with the minimum travel time and



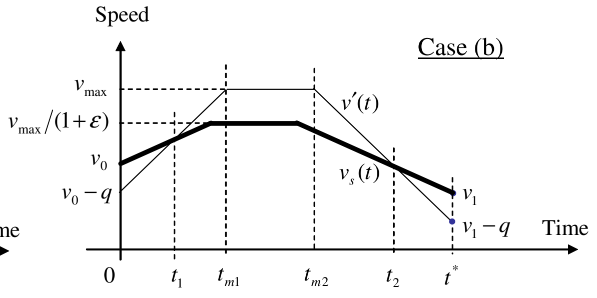


Figure 5.6: Time-speed profiles for the trajectories with maximum travel distance: (a) $v'(t) < v_{max}$, (b) $v'(t) = v_{max}$ for some t

Γ' with the maximum travel distance. Hereafter, with a little abuse of notation, we use Γ_s and Γ' to represent such trajectories: i.e., $\Gamma_s = \arg\min_{\Gamma_s} |\Gamma_s|$ and $\Gamma' = \arg\max_{\Gamma'} l'$.

Let $v_s(t), v'(t)$ denote the time-speed profiles of Γ_s and Γ', respectively. We can divide the time interval $[0, t^*]$ into three intervals according to the relationship between two time-speed profiles as follows: 1) $[0, t_1]$: $v_s(t) \geq v'(t)$, 2) $[t_1, t_2]$: $v_s(t) \leq v'(t)$, and 3) $[t_2, t^*]$: $v_s(t) \geq v'(t)$.

We first consider how much the trajectory Γ_s lags behind Γ' in the intervals $[0, t_1]$ and $[t_2, t^*]$. Since $1 - 1/(1+\epsilon)^2 > \epsilon/2$ for $0 < \epsilon < 1$, we can show

$$t_1 = \frac{q_0}{a_{max} - \frac{a_{max}}{(1+\epsilon)^2}} < \frac{2q}{\epsilon a_{max}}, \qquad t^* - t_2 = \frac{q_1}{a_{max} - \frac{a_{max}}{(1+\epsilon)^2}} < \frac{2q}{\epsilon a_{max}}. \tag{5.20}$$

Then we have

$$\int_0^{t_1} (v_s(t) - v'(t))dt < \frac{q^2}{\epsilon a_{max}}, \qquad \int_{t_2}^{t^*} (v_s(t) - v'(t))dt < \frac{q^2}{\epsilon a_{max}}. \tag{5.21}$$

Therefore, to satisfy $\max\{l'\} \geq l$, it is sufficient to choose q that guarantees

$$\int_{t_1}^{t_2} (v'(t) - v_s(t))dt \;\; \geq \;\; \frac{2q^2}{\epsilon a_{max}}. \tag{5.22}$$

We consider sufficient conditions to satisfy (5.22) separately for the case of $0 \leq \forall t \leq t^*.\ v'(t) < v_{max}$ and the case of $0 \leq \exists t \leq t^*.\ v'(t) = v_{max}$.

First we consider the case of $v'(t) < v_{max}$ in $[t_1, t_2]$ (Figure 5.6, Case (a)[5]). Then, for $t_1 < t \leq t_m$,

$$v'(t) - v_s(t) \;\; \geq \;\; \left(a_{max} - \frac{a_{max}}{(1+\epsilon)^2}\right)(t - t_1) \;\; > \;\; \frac{\epsilon a_{max}}{2}(t - t_1). \tag{5.23}$$

[5]The figure shows typical cases only. Specifically, it is possible that $v_s(t)$ reaches $v_{max}/(1+\epsilon)$ and keeps that speed before going down to v_1. However, the analysis holds for such cases as well.

Similarly for $t_m \leq t < t_2$,

$$v'(t) - v_s(t) \quad > \quad \frac{\epsilon a_{max}}{2}(t_2 - t). \tag{5.24}$$

Using (5.23) and (5.24),

$$\begin{aligned}\int_{t_1}^{t_2} (v'(t) - v_s(t))dt &= \int_{t_1}^{t_m} (v'(t) - v_s(t))dt + \int_{t_m}^{t_2} (v'(t) - v_s(t))dt \\ &> \frac{\epsilon a_{max}}{2}\left(t_m^2 - (t_1 + t_2)t_m + \frac{t_1^2 + t_2^2}{2}\right) \\ &\geq \frac{\epsilon a_{max}}{8}(t_2 - t_1)^2. \end{aligned} \tag{5.25}$$

Then a sufficient condition to satisfy (5.22) is

$$\frac{\epsilon a_{max}}{8}(t_2 - t_1)^2 \geq \frac{2q^2}{\epsilon a_{max}}. \tag{5.26}$$

Using $t_2 - t_1 > t^* - 4q/\epsilon a_{max}$ (from (5.20)), a sufficient condition to satisfy (5.26) is

$$\frac{4q}{\epsilon a_{max}} \leq t^* - \frac{4q}{\epsilon a_{max}}, \tag{5.27}$$

which is equivalent to $q \leq \epsilon a_{max} t^*/8$. Since $t^* \geq l/v_{max}$, we have

$$q \leq \frac{\epsilon}{8} \frac{a_{max} l}{v_{max}}. \tag{5.28}$$

Now we consider the other case. As shown in Figure 5.6 (Case (b)[6]), there is a time interval $[t_{m1}, t_{m2}]$ where $v'(t) = v_{max}$. For $t_{m1} \leq t \leq t_{m2}$, using $\epsilon/(1+\epsilon) > \epsilon/2$ for $0 < \epsilon < 1$,

$$v'(t) - v_s(t) \quad \geq \quad v_{max} - \frac{v_{max}}{1+\epsilon} \quad > \quad \frac{\epsilon v_{max}}{2}. \tag{5.29}$$

Since $v'(t_{m1}) = v_{max}$ and $v'(t_1) \leq v_{max}/(1+\epsilon)$,

$$t_{m1} - t_1 \quad \geq \quad \frac{v_{max} - \frac{v_{max}}{1+\epsilon}}{a_{max}} \quad > \quad \frac{\epsilon v_{max}}{2a_{max}}, \tag{5.30}$$

$$t_2 - t_{m2} \quad > \quad \frac{\epsilon v_{max}}{2a_{max}}. \tag{5.31}$$

Then, using $1 - 1/(1+\epsilon)^2 > \epsilon/2$ for $0 < \epsilon < 1$,

$$\begin{aligned}\int_{t_1}^{t_2} (v'(t) - v_s(t))dt &= \left\{\int_{t_1}^{t_{m1}} + \int_{t_{m1}}^{t_{m2}} + \int_{t_{m2}}^{t_2}\right\}(v'(t) - v_s(t))dt \\ &> \frac{\epsilon a_{max}}{4}\left((t_{m1} - t_1)^2 + (t_2 - t_{m2})^2\right) + \frac{\epsilon v_{max}}{2}(t_{m2} - t_{m1}) \\ &> \frac{\epsilon^3 v_{max}^2}{8a_{max}}. \end{aligned} \tag{5.32}$$

[6]Again, the figure shows typical cases only. There are more variations for this case, such as $v_s(t)$ reaches $v_{max}/(1+\epsilon)$ before intersecting with $v'(T)$, or $v_s(t)$ does not reach $v_{max}/(1+\epsilon)$. For all possible cases, the analysis is valid.

To guarantee that (5.22) is satisfied, a sufficient condition is

$$\frac{\epsilon^3 v_{max}^2}{8a_{max}} \geq \frac{2q^2}{\epsilon a_{max}} \tag{5.33}$$

and thus

$$q \leq \frac{\epsilon^2 v_{max}}{4}. \tag{5.34}$$

From (5.28) and (5.34), the lemma follows. □

Based on Lemma 5.3.10, we choose the speed quantization step q_n at each dominant point as follows:

$$q_n \quad = \quad \min\left\{\frac{\epsilon}{8}\frac{a_{max}}{v_{max}}\min_i\{x_{i+1}-x_i\}, \frac{\epsilon^2 v_{max}}{4}\right\}. \tag{5.35}$$

To summarize, for any $(a_{max}/(1+\epsilon)^2, v_{max}/(1+\epsilon))$-bounded trajectory, there exists an (a_{max}, v_{max})-bounded trajectory that travels the same distance and, at each endpoint, the speed is at most that of the original one and an integer multiple of q_n.

Short Intervals

The speed quantization step derived in (5.35) depends on $\min_i\{x_{i+1}-x_i\}$, the minimum difference between two neighboring dominant points. This is not preferable because the number of possible speed choices will be very large when the difference is very small, resulting in indefinitely long running time of the algorithm based on dynamic programming.

For this reason, we handle such short intervals differently. In normal intervals, we have determined the speed quantization step so that: 1) there exists an (a_{max}, v_{max})-bounded discrete trajectory that takes the same amount of time in each interval as an $(a_{max}/(1+\epsilon)^2, v_{max}/(1+\epsilon))$-bounded continuous trajectory, and 2) the speed of the discrete trajectory at each dominant point is immediately below that of the continuous trajectory. In short intervals, however, we relax both of these conditions and change them as follows: 1') in each short interval, the discrete trajectory takes at most Δ_t time more than the continuous trajectory does, and 2') the speed of the discrete trajectory at each dominant point is in a certain range below that of the continuous trajectory.

We consider an interval short when $q_n < \epsilon^2 v_{max}/4$ The choice of this threshold is arbitrary, but we choose this value for simplicity. From (5.35), an interval is short if the length l satisfies $l < 2\epsilon v_{max}^2/a_{max}$.

First we determine the quantization step q_s and the speed range so that there always exists a discrete trajectory corresponding to any continuous trajectory. The problem of indefinitely long running time can be overcome by allowing a discrete trajectory to take a quantized speed that is not immediately below the speed of the continuous one. As we see later, this results in multiple speed choices at each dominant point. In addition, on transition from a short interval to a normal interval, the whole speed range must satisfy the condition in Lemma 5.3.10. The following lemma gives a speed range that satisfies all these conditions:

Lemma 5.3.11. *For arbitrary $(a_{max}/(1+\epsilon)^2, v_{max}/(1+\epsilon))$-bounded trajectory Γ_1 from $(l,v)=(0,v_0)$ to (l,v_1) and any constants $q_s>0$ and integer m_0 that satisfy the following constraints for any $k\in\{0,1,2,\ldots,2n\}$:*

$$\begin{cases} \max\{v_0/q_s-\alpha_k,0\}\le m_0\le v_0/q_s-\beta_k & \textit{If } v_0-\beta_k q_s\ge 0 \\ m_0=0 & \textit{Otherwise,} \end{cases} \tag{5.36}$$

where $\alpha_k\equiv 2n+k+1$ and $\beta_k\equiv 2n-k$, there exists an (a_{max},v_{max})-bounded trajectory Γ_2 from $(0,m_0q_s)$ to (l,m_1q_s) where m_1 is an integer satisfying

$$\begin{cases} \max\{v_1/q_s-\alpha_{k+1},0\}\le m_1\le v_1/q_s-\beta_{k+1} & \textit{If } v_1-\beta_{k+1} q_s\ge 0 \\ m_1=0 & \textit{Otherwise.} \end{cases} \tag{5.37}$$

Proof. We assume $v_0-\alpha_k q_s\ge 0$ and $v_1-\alpha_{k+1}q_s\ge 0$, since all other cases are easily shown from the result for this case. For any m_0q_s, if $|m_0q_s-m_1q_s|\le|v_0-v_1|$, then transition to m_1q_s is possible. For a speed range $[v_0-pq_s, v_0-(p+1)q_s]$, consider a corresponding speed range $[v_1-pq_s, v_1-(p+1)q_s]$. Assume m_0q_s is in the first range and $m_0'q_s$ is in the corresponding range. If $|m_0q_s-m_0'q_s|\le|v_0-v_1|$ then just set $m_1=m_0'$, since the transition from m_0q_s to $m_0'q_s$ is clearly possible. Otherwise, since either $|m_0q_s-(m_0'+1)q_s|\le|v_0-v_1|$ or $|m_0q_s-(m_0'-1)q_s|\le|v_0-v_1|$ is satisfied, set $m_1=m_0'+1$ or $m_0'-1$ accordingly. Since $v_0-\alpha_kq_s\le m_0q_s\le v_0-\beta_kq_s$, we have $v_1-(\alpha_k+1)q_s\le m_1q_s\le v_1-(\beta_k-1)q_s$. □

Figure 5.7 explains the idea of Lemma 5.3.11. For a dominant point between normal intervals (x_0), a speed-quantized trajectory takes the speed in the range $[v_0-q_n, v_0]$ for quantization step q_n determined by Eq. (5.35). The vertical interval in the figure shows the speed range that a speed-quantized trajectory can take, and thus the quantization step must be at most the length of the speed range to guarantee that there

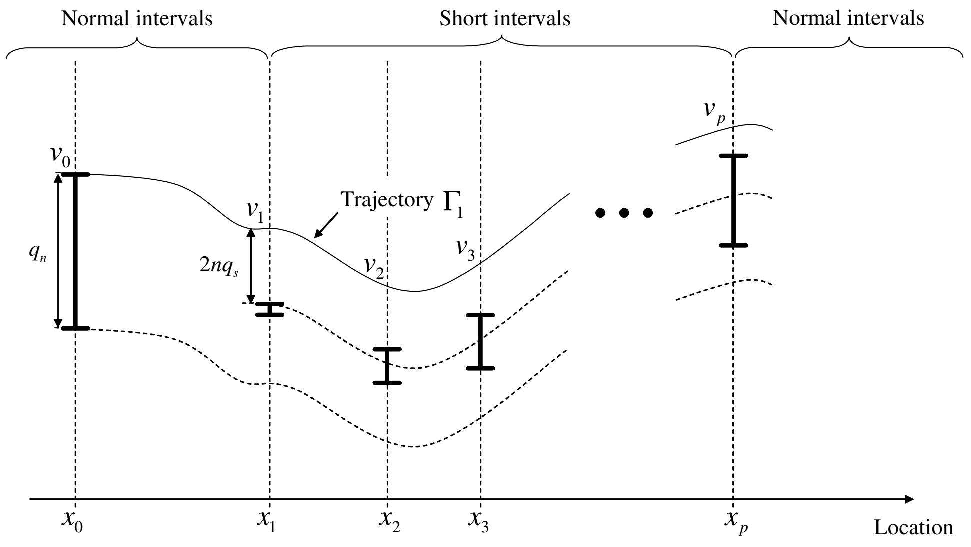


Figure 5.7: Example of consecutive short intervals and transitions to/from normal intervals: Bold lines show the speed range that a speed-quantized trajectory can take.

is at least one speed point in the range. On the boundary of normal and short intervals (x_1), speed is discretized by step q_s and speed range is $[v_1-(2n+1)q_s, v_1-2nq_s]$. For this to be possible, q_s must satisfy $(2n+1)q_s < q_n$ (from Lemma 5.3.10), but this condition can be ignored since it is looser than the one imposed at the transition from a short interval to a normal one. When there are consecutive short intervals, the speed range grows as shown at x_2, x_3, and x_p.

We have a condition on q_s imposed when moving from a short interval to a normal interval. At x_p in Figure 5.7, when the sequence of short intervals ends, we need to guarantee that 1) the top of the speed range is equal to or less than v_p, and 2) the bottom of that is the equal to or more than $v_p - q_n$. The condition 1 is always satisfied because there are at most $2n$ consecutive short intervals before a normal interval. We need to satisfy the condition 2, which is equivalent to $(2n+2n+1)q_s \leq \epsilon^2 v_{max}/4$.

Then we consider an upper bound on the additional time that discrete trajectory takes in each short interval.

Lemma 5.3.12. *In Lemma 5.3.11, there exists* Γ_2 *satisfying* $|\Gamma_1| \leq |\Gamma_2| \leq |\Gamma_1| + (4n + 2k+3)q_s/a_{max}$.

Proof. Since the speed of Γ_2 at dominant points are equal to or less than that of Γ_1, it is possible for Γ_2 to travel the distance equal to or less than l in time $|\Gamma_1|$. We consider another trajectory that consists of following three steps: 1) Accelerate from $m_0 q_s$ to v_0 at the rate a_{max}, 2) Emulate Γ_1, 3) Decelerate from v_1 to $m_1 q_s$ at the rate a_{max}. This trajectory clearly travels at least l and the total additional time in step 1 and 3 is at most $(2n+k+1)q_s/a_{max} + (2n+k+2)q_s/a_{max}$.

In the rest of the proof we use the notation $L_{max}(t) \equiv L_{max}(a_{max}, v_{max}, t, m_0 q_s, m_1 q_s)$. We also define $L_{min}(t)$ similarly. From the above discussion, we have $L_{min}(|\Gamma_1|) \leq l$ and $L_{max}(|\Gamma_1| + \delta) \geq l$, where we define $\delta = (4n+2k+3)q_s/a_{max}$ for clarity. If $L_{max}(|\Gamma_1|) \geq l$ or $L_{min}(|\Gamma_1| + \delta) \leq l$, the lemma follows from Lemma 5.3.7. Otherwise, we have $L_{max}(|\Gamma_1|) < l$. For this case, since we already have $L_{max}(|\Gamma_1| + \delta) \geq l$ and $L_{max}(t)$ is an increasing function for t, there exists t' such that $|\Gamma_1| \leq t' \leq |\Gamma_1| + \delta, L_{max}(t') = l$ and thus the lemma follows. □

We want the total additional time to be bounded by $\epsilon|\Gamma_{OPT}|$:

$$\sum_{k=1}^{2n+1} \frac{(4n+2k+3)q_s}{a_{max}} \quad \leq \quad \epsilon|\Gamma_{OPT}|. \tag{5.38}$$

Since $|\Gamma_{OPT}| \geq 2\sqrt{L/a_{max}}$, where the right hand side is the fastest possible travel time from $(0,0)$ to $(L,0)$, a sufficient condition for (5.38) is $q_s \leq \epsilon\sqrt{a_{max}L}/(2n+1)(6n+5)$.

To summarize, we choose the speed quantization step of short intervals as follows:

$$q_s \quad = \quad \min\left\{\frac{\epsilon\sqrt{a_{max}L}}{(2n+1)(6n+5)}, \frac{\epsilon^2 v_{max}}{4(4n+1)}\right\} \tag{5.39}$$

Note that it is possible to treat all intervals as short and quantize the speed by q_s. However, since q_s is smaller than the step of normal intervals by at least a factor of n, thus results in unnecessarily long running time.

Procedure for choosing the speed quantization step

Combining the results for normal intervals and short intervals, we choose the speed quantization step q_i at dominant point x_i as follows:

$$q_i = \begin{cases} \min\left\{\frac{\epsilon\sqrt{a_{max}L}}{(2n+1)(6n+5)}, \frac{\epsilon^2 v_{max}}{4(4n+1)}\right\} & \text{If } \min\{x_i - x_{i-1}, x_{i+1} - x_i\} \leq \frac{2\epsilon v_{max}^2}{a_{max}} \\ \\ \frac{\epsilon^2 v_{max}}{4} & \text{Otherwise} \end{cases} \tag{5.40}$$

The following lemma states the existence of a trajectory that is $(1+\epsilon)$-stretched feasible and speed-quantized:

Lemma 5.3.13. *Let ϵ be a small constant satisfying $0 < \epsilon < 1$. For any $(a_{max}/(1+\epsilon)^2, v_{max}/(1+\epsilon))$-bounded trajectory Γ_s from $(0,0)$ to $(L,0)$, there exists an (a_{max}, v_{max})-bounded trajectory Γ_q from $(0,0)$ to $(L,0)$ that satisfies 1) the time staying in the location interval $[x_i, x_{i+1}]$ is at least as long as that in Γ_s, 2) the speed at dominant point x_i is equal to or less than that of Γ_s and an integer multiple of q_i, where q_i is given by Eq. (5.40), and 3) $|\Gamma_q| \le |\Gamma_s| + \epsilon|\Gamma_{OPT}|$.*

Proof. If there are no short intervals in the whole travel interval $[0, L]$, from Lemma 5.3.10, Conditions 1) and 2) are clearly satisfied and Condition 3) is also satisfied because $|\Gamma_q| \le |\Gamma_s|$.

When there is a short interval, Conditions 1) and 2) are satisfied from Lemma 5.3.11. Condition 3) is also satisfied since the additional travel time is upper bounded by Lemma 5.3.12 and the quantization step for short intervals is determined such that the total additional time is upper bounded by $\epsilon|\Gamma_{OPT}|$ (in Eq. (5.38)). □

5.3.4 Time Quantization

Having quantized the speed, we now quantize the cumulative execution time at each dominant point. Note that the travel time of data mule is not quantized. The following lemma gives a constraint on the step size so that the feasibility is guaranteed after the quantization:

Lemma 5.3.14. *Let ϵ be a small constant satisfying $0 < \epsilon < 1$. For trajectory Γ_s that is $(1+\epsilon)$-stretched feasible for $\mathcal{J}$, let $t_{\Gamma_s}(x)$ denote a $(1+\epsilon)$-stretched feasible cumulative execution time function. Then, there exists another cumulative execution time function $t'_{\Gamma_s}(x)$ that is (1-stretched) feasible and the value at each dominant point is an integer multiple of τ, where τ satisfies $\tau|e_i$ for all i and $\tau \le \{\epsilon/(2n+1)\}\min_i\{e_i\}$, where n is the number of location jobs in $\mathcal{J}$ and "$a|b$" means that b is an integer multiple of a.*

Proof. Let x_i, x_{i+1} denote neighboring dominant points. We can construct $t'_{\Gamma_s}(x)$ inductively as $t'_{\Gamma_s}(x_{i+1}) = t'_{\Gamma_s}(x_i) + \tau\lfloor(t_{\Gamma_s}(x_{i+1}) - t_{\Gamma_s}(x_i))/\tau\rfloor$. The last term is the delta of $t_{\Gamma_s}(x)$ rounded down to an integer multiple of τ. Then $t'_{\Gamma_s}(x)$ is a cumulative execution time function for Γ_s, since we can always reduce the execution time by considering idle time. Next we consider a condition for τ so that $t'_{\Gamma_s}(x)$ is (1-stretched) feasible. For each interval, allocated time in $t'_{\Gamma_s}(x)$ is equal to or less than that in $t_{\Gamma_s}(x)$ by at most τ. Since there are up to $(2n+1)$ location intervals, total reduction of allocated time for each job is at most $(2n+1)\tau$. To maintain the feasibility, the allocation time for each job must be at least its execution time. As $t_{\Gamma_s}(x)$ is $(1+\epsilon)$-stretched feasible, we need $(1+\epsilon)e_j - (2n+1)\tau \geq e_i$ for job i. The lemma follows. □

5.3.5 Dynamic Programming

We use a three-dimensional table of (location, speed, cumulative execution time). The entry in (i, j, k) is the travel time of the fastest feasible speed-quantized trajectory from $(l, v) = (0, 0)$ to (x_i, jq_i) that has cumulative execution time $k\tau$ at location x_i. The table is initialized by 0 at $(0, 0, 0)$ and $+\infty$ otherwise. The final output is the entry in $(N_d - 1, 0, \sum e_m)$, where N_d is the number of dominant points. This is the minimum travel time of the feasible trajectory from $(0, 0)$ to $(L, 0)$ of which the speed and cumulative execution time are quantized at each dominant point.

The table is updated in the following way along the location index. There are two types of transitions. A transition from (i, j, k) to $(i+1, j', k')$, which we call a *type-I transition*, is possible when all of the following constraints are satisfied. We define $v = jq_i$ and $v' = j'q_{i+1}$.

- (Valid movement) $|v'^2 - v^2| \leq 2a_{max}(x_{i+1} - x_i)$.
- (Feasibility) If $j' > 0$, $\sum_{m:d_m \leq x_{i+1}} e_m \leq k'\tau \leq \sum_{m:r_m \leq x_{i+1}} e_m$. From Lemma 5.3.6, this constraint is equivalent to the feasibility constraint. The right inequality means that the cumulative execution time cannot exceed the sum of execution time of all released jobs.

  Note that this constraint is not necessary when $j' = 0$, since the data mule can stop at x_{i+1} and execute the jobs ("type-II transition," which is discussed later). For such cases, we have the following constraint on departing from a location: if $j = 0$, $\sum_{m:d_m \leq x_i} e_m \leq k\tau \leq \sum_{m:r_m \leq x_i} e_m$.
- (Cumulative execution time) $k'\tau - k\tau \leq T_{max}(a_{max}, v_{max}, x_{i+1} - x_i, v, v')$. This is because the increase of cumulative execution time must be equal to or less than the travel time. As the data mule can be idle while moving, we do not need the lower bound condition.

When all these conditions are met, we can update the entry at $(i+1, j', k')$. Let T and T' denote the current entry at (i, j, k) and $(i+1, j', k')$, respectively. We update the entry by $T + \max(k'\tau - k\tau, T_{min}(a_{max}, v_{max}, x_{i+1} - x_i, v, v'))$ if this is less than T'.

When $j = 0$, a transition from $(i, 0, k)$ to $(i, 0, k')$ $(k < k')$ is also possible. We call this a *type-II transition.* This corresponds to the case that the data mule stops at x_i to execute the jobs. In this case, k' needs to satisfy only the feasibility condition: $\sum_{m:d_m \le x_i} e_m \le k'\tau \le \sum_{m:r_m \le x_i} e_m$.

As for the order of updating, first we must do all the type-II transitions and then the type-I transitions later so that all possible transitions are explored.

5.3.6 Proof of Theorem 5.3.1

We first construct a slower trajectory from Γ_{OPT} and then convert it to a quantized version without violating the feasibility. By Lemma 5.3.9, we can convert Γ_{OPT} into a $(a_{max}/(1+\epsilon)^2, v_{max}/(1+\epsilon))$-bounded trajectory Γ_s that is $(1+\epsilon)$-stretched feasible. Note that $|\Gamma_s| = (1+\epsilon)|\Gamma_{OPT}|$. From Lemma 5.3.13, there exists a (a_{max}, v_{max})-bounded discrete trajectory Γ_1 that spends at least the same amount of time as Γ_{OPT} does for each location interval, and that the additional time for whole travel interval is at most $\epsilon|\Gamma_{OPT}|$. The speed at each dominant point is equal to or less than that of Γ_s and is an integer multiple of q_i (from Eq. (5.40)).

Since Γ_1 is $(1+\epsilon)$-stretched feasible, there exists a cumulative execution time function $t'_{\Gamma_1}(x)$ that is (1-stretched) feasible and is discretized by step τ determined by Lemma 5.3.14. From Lemma 5.3.13, Γ_1 is a trajectory from $(0, 0)$ to $(L, 0)$, as the speed of Γ_1 at each dominant point is equal or less than that of Γ_{OPT}.

Let N_s and N_c denote the number of quantization steps of speed and cumulative execution time. From Lemma 5.3.13 and Lemma 5.3.14,

$$N_s \le \frac{v_{max}}{q_i} = O\left(n^2\left(\frac{1}{\epsilon}\right)^2 \frac{v_{max}}{\sqrt{a_{max}L}}\right), \tag{5.41}$$

$$N_c \le \frac{\sum e_i}{\tau} = \frac{\sum e_i}{\frac{\epsilon}{2n+1}\min_i\{e_i\}} = O\left(n\left(\frac{1}{\epsilon}\right)\frac{\sum e_i}{\min_i\{e_i\}}\right). \tag{5.42}$$

For each neighboring pair of dominant points, there are at most $(N_s N_c)^2$ type-I transitions. There are at most N_c^2 type-II transitions for each dominant point. Each update is done in constant time. Since there are at most $2n + 2$ dominant points, the running time is at most $(2n + 2)(N_s^2 N_c^2 + N_c^2)$, which leads to (5.3).

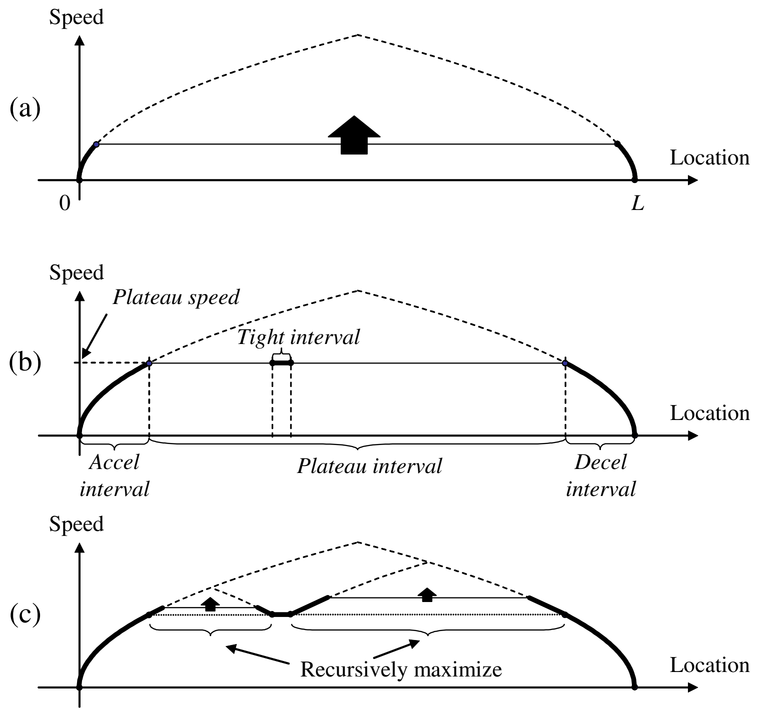


Figure 5.8: Idea of the proposed heuristic algorithm: (a) Increase the plateau speed. Two dotted curves show the acceleration/deceleration for the fastest possible travel covering the whole interval. Bold lines and thin lines mean fixed intervals and free intervals, respectively. (b) A tight interval is found. New terms are shown in italic. (c) Recursively maximize the speed for the remaining free intervals.

5.4 Heuristic Algorithm

We design an efficient heuristic algorithm for Generalized 1-D DMS problem that works for both simple and general location jobs cases. This algorithm always returns a "feasible" speed control plan, i.e., the induced job scheduling problem has a feasible schedule. We first present the idea of the algorithm and then explain the details, followed by discussions and some possible extensions.

5.4.1 Overview of Approach

Figure 5.8 shows the idea of the heuristic algorithm. We first design the heuristic algorithm for the case where the speed is constrained to be zero at 0 and L, and extend it to an unconstrained case later. The algorithm works recursively, and in each recursion, we confine ourselves to the following 3-phase speed change profile: first accelerate at the maximum acceleration, then move at the constant speed, and finally decelerate at the maximum negative acceleration. We call each of these intervals *accel interval*, *plateau interval*, and *decel interval*, respectively. All of these are actually location intervals, but we omit "location" for simplicity. Further we call the speed in the plateau interval as *plateau speed*.

As shown in Figure 5.8(b), the main idea of the algorithm is to maximize the plateau speed until we have a *tight interval*, defined as an interval whose length (in time) is equal to the processor demand for that interval. Processor demand for an interval is defined as the sum of execution time of the jobs whose feasible interval is completely contained in the interval. We can naturally extend the definition to define *tight location interval*, when we give the speed of data mule for the interval. We simply call it a tight interval as well.

For convenience, we define the terms *fixed interval* and *free interval*. Fixed intervals are the location intervals where the speed is already maximized. Accel interval, decel interval, and tight interval are all fixed intervals. The remaining intervals, specifically the intervals in plateau interval but not in tight interval, are called free intervals. We can further increase the speed for free intervals without destroying the feasibility.

The heuristic algorithm recursively applies maximization to the free intervals, as shown in Figure 5.8(c). The algorithm terminates when there is no free interval.

After determining the speed control plan, finding a feasible job schedule is easy: we can use the EDD algorithm, which is possible because the heuristic algorithm first converts all general location jobs to simple location jobs.

5.4.2 Algorithm Details

The heuristic algorithm consists of four steps: simplify, maximize, trim, and recursion.

Simplify

The procedure Simplify-Jobset converts all general location jobs to simple location jobs. For each general location job having k feasible location intervals, we create k simple location jobs. The execution time is distributed to each feasible location interval proportionally to its length. If all the feasible location intervals of a general location job are zero-length, we just distribute the execution time equally to them.

```
1: procedure SIMPLIFY-JOBSET($\mathcal{J}$)
2:     for each general location job $\tau \in \mathcal{J}$ do
3:         if $\sum_{I \in \mathcal{I}(\tau)} |I| > 0$ then
4:             for each feasible location interval $I \in \mathcal{I}(\tau)$ do
5:                 $\mathcal{J}' \leftarrow \mathcal{J}' \cup \left\{ I, \frac{|I|}{\sum_{I \in \mathcal{I}(\tau)} |I|} e(\tau) \right\}$
6:             end for
7:         else
8:             for each feasible location interval $I \in \mathcal{I}(\tau)$ do
9:                 $\mathcal{J}' \leftarrow \mathcal{J}' \cup \left\{ I, \frac{1}{N(\tau)} e(\tau) \right\}$
10:            end for
11:        end if
12:    end for
13:    return $\mathcal{J}'$
14: end procedure
```

$N(\tau)$ (in Line 9) is the number of feasible location intervals of general location job τ. In Line 4 and 8, a location job is denoted as {(set of feasible location intervals), (execution time)}.

Maximize

In this step, we maximize the plateau speed until we have a tight interval. For every possible location interval from the release location of one job and the deadline location of another job, calculate the speed that makes it a tight interval. We choose the plateau speed by finding the minimum of them for all the intervals. This procedure does not destroy the feasibility, since the processor demand for every location interval is equal or less than the time allocated for that interval.

Given a set $\mathcal{J}$ of location jobs, a baseline interval I_0, and a baseline speed v_0, the procedure MAXIMIZE finds the plateau speed. We provide baseline interval and speed as the input, since we use this procedure recursively, as explained later.

```
1: procedure MAXIMIZE($\mathcal{J}, I_0, v_0$)
2:     for each location interval $I = [r(\tau), d(\tau')]$ s.t. $\tau, \tau' \in \mathcal{J}$, $r(\tau) \leq d(\tau')$ do
3:         $u(I) \leftarrow$ MAX-SPEED($\mathcal{J}, I$)
4:     end for
5:     $v_p \leftarrow \min_I \{u(I)\}$                                    ▷ Plateau speed
```

Higher $\longleftarrow$ Processor demand: $PD(I)$ $\longrightarrow$ Lower

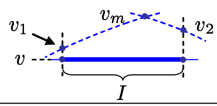
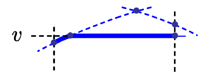
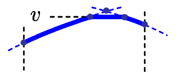
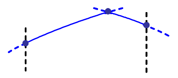
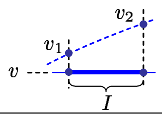
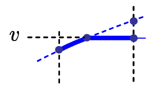
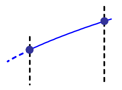
	Case 1: $v \le v_a$	Case 2: $v_a < v \le v_b$	Case 3: $v_b < v \le v_m$	Case 4: $v_m < v$
(A) Different side				
(B) Same side				
Maximum plateau speed	$v = \dfrac{\lvert I\rvert}{PD(I)}$	$v = \alpha - \sqrt{\alpha^2 - \beta}$ $\alpha = v_a + a_{max} \cdot PD(I)$ $\beta = 2a_{max} \cdot \lvert I\rvert + v_a^2$	$v = \frac{1}{2}\left(\alpha - \sqrt{\alpha^2 - 2\beta}\right)$ $\alpha = (v_1 + v_2) + a_{max} \cdot PD(I)$ $\beta = 2a_{max} \cdot \lvert I\rvert + (v_1^2 + v_2^2)$	$v = +\infty$

Figure 5.9: Determining maximum plateau speed: v_1, v_2 are the maximum possible speed at $l[I], h[I]$ that are determined by the accel/decel curves. We define $v_a = \min\{v_1, v_2\}$, $v_b = \max\{v_1, v_2\}$, and v_m as the maximum possible speed within interval I assuming no feasibility constraints. Only the case of $v_1 < v_2$ is shown, but the same equations hold for $v_1 \ge v_2$ case as well.

```
6:      **if** $v_p \neq +\infty$ **then**
7:          $I_{tight} \leftarrow \arg\min_I(u(I))$
8:          $I_{accel} \leftarrow [l[I_0],\ l[I_0] + (v_p^2 - v_0^2)/2a_{max}]$
9:          $I_{decel} \leftarrow [h[I_0] - (v_p^2 - v_0^2)/2a_{max},\ h[I_0]]$
10:     **else**
11:         $I_{tight} \leftarrow \emptyset$
12:         $I_{accel} \leftarrow [l[I_0],\ (l[I_0] + h[I_0])/2]$
13:         $I_{decel} \leftarrow [(l[I_0] + h[I_0])/2,\ h[I_0]]$
14:     **end if**
15:     Trim and Recursion
16: **end procedure**
```

$l[I]$ and $h[I]$ denote the lower and higher endpoints of interval I, respectively. Max-Speed (in Line 3) is a procedure to find the maximum speed that preserves the feasibility for location interval I.

In Max-Speed, the maximum plateau speed is determined according to the table shown in Figure 5.9. It is done by analyzing the processor demand for I and computing the speed that makes I a tight interval. Each graph shows the speed change when the maximum plateau speed is realized. Two dotted curves in each graph represent the maximum acceleration from and deceleration to the start/end of the baseline interval. Let's call them “accel curve” and “decel curve”, respectively. Any speed changes cannot exceed the accel/decel curves and thus maximum possible speed at each location is determined by these curves. There are two cases depending on the relation between interval I and the accel/decel curves. First is the case that, at the start of I, the accel curve is below the decel curve (i.e., maximum speed is determined by the accel curve) and it is the opposite at the end of I. This is shown as the “different side” case in the figure. The other is the “same side” case, in which the accel curve is above (or below) the decel curve at both endpoints of I and thus the maximum speed is determined solely by the accel (or decel) curve.

In the table, Case 1 to 4 show different speed changes depending on the processor demand for I, which is the highest in Case 1 and lowest in Case 4. To determine which case applies, we first assume that Case 1 applies and calculate the maximum plateau speed by using the equation shown in the table. Then, if the result satisfies the constraint (e.g., $v \leq v_a$ in Case 1), it means that Case 1 actually applies and we found the maximum plateau speed. Otherwise, we move on to Case 2 and do the same thing until we find the case in which the constraint is satisfied. Note that maximum plateau speed is infinity in Case 4 and so the procedure eventually exits.

Trim

After identifying the tight interval and the corresponding plateau speed, we trim the feasible location interval of each location job. By trimming, we can make all the location jobs completely contained in the free intervals, and thus we can further increase the speed in these intervals independently from other fixed intervals.

We first consider the plateau speed $v_p > 0$ case. Figure 5.10 explains the details of trimming. For the accel interval, we simulate the EDD algorithm to see how much time is allocated to each job within the interval. It is possible because we can map each location point to a time point using the knowledge that the initial speed is v_0 and the acceleration is a_{max}. Let a_i denote the total allocated time to job τ_i at the end of the accel interval. A job completely contained in the interval (τ_1) will be finished in the interval (i.e., $a_1 = e_1$). For a job that intersects with the interval (τ_2), we trim off the feasible location interval at the end of the accel interval, and decrease the execution time to $e_2 - a_2$.

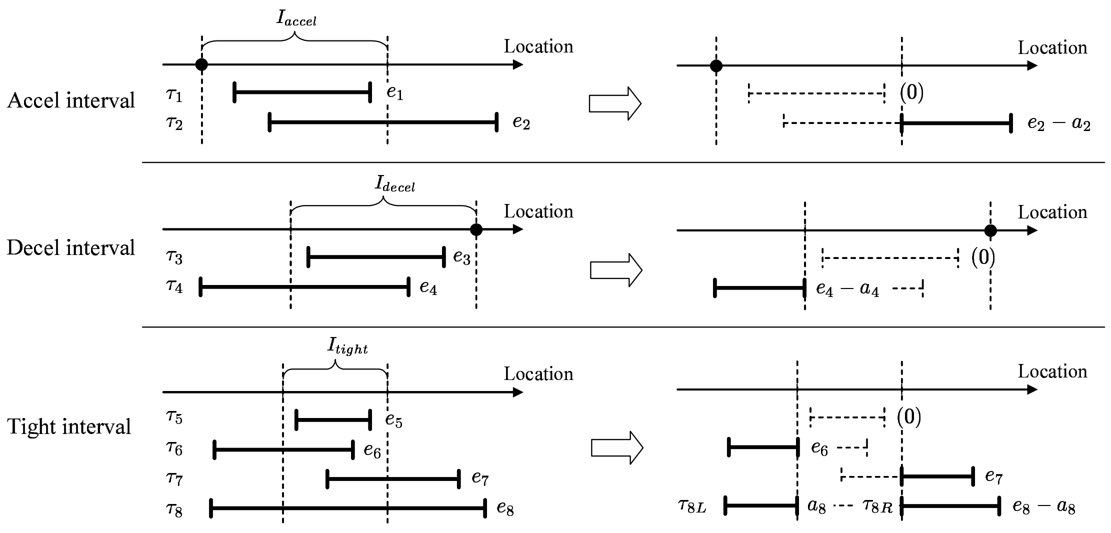


Figure 5.10: Trimming feasible location intervals: accel, decel, and tight intervals are trimmed off from the feasible location interval of each location job. Execution time is shown on the right of each interval.

For the decel interval, it works in the same way except that we simulate the LRT (latest release time) algorithm instead of the EDD algorithm. The LRT algorithm schedules jobs backwards, treating release times as deadlines, and is also known to be optimal [Liu00].

The tight interval is processed differently. In a tight interval, by definition, time is allocated only to the jobs that are completely contained in the interval (τ_5). Thus, for the jobs that intersect a tight interval but not completely contained (τ_6 and τ_7), we trim off the feasible location interval and their execution times are unchanged. For a job that completely contains the tight interval (τ_8), we divide it into two jobs (τ_{8L} and τ_{8R}), one before the tight interval and the other after that. How to distribute the execution time to these two jobs without destroying the feasibility is not a trivial problem. As one of the simplest ways, we simulate the EDD algorithm for the location interval from the end of the accel interval to the beginning of the tight interval[7]. Assuming time a_8 is allocated to job τ_8 within that interval, we assign the execution time a_8 to job τ_{8L} and $e_8 - a_8$ to τ_{8R}. When a_8 is zero, τ_{8L} is not created. Similarly for τ_{8R}.

When the plateau speed $v_p = 0$, there is at least one simple location job having a zero-length feasible location interval. Then, each of accel, decel, and tight intervals

[7]Alternatively we could simulate the LRT algorithm for the location interval from the end of the tight interval to the end.

degenerates to a point. In this case, we only divide each of the jobs that straddle the point into two. Execution time is proportionally distributed to two newly created jobs according to the length of feasible location intervals.

Recursion

At this point, if there are any remaining jobs, the feasible location intervals of these jobs are completely contained in the free intervals that are neither accel, decel, nor tight intervals. For these free intervals, since there may be some room for increasing the speed, we recursively maximize the speed by repeating from the maximization procedure. As long as we maintain the feasibility condition in these intervals, the entire speed control plan remains feasible because of the trimming procedure we use.

Figure 5.11 shows the possible cases of recursion and corresponding output of speed control plan. The number of recursions is zero (Case 0-1, 0-2), one (Case 1-1, 1-2), or two (Case 2), depending on the configuration of the tight interval. Here each speed control plan is represented as a collection of acceleration segments. An acceleration segment s is represented as a tuple $\{I(s), z(s), v(s), a(s)\}$, where $I(s)$ is the location interval, $z(s)$ is the time duration, $v(s)$ is the initial speed at $l[I(s)]$, and $a(s)$ is the acceleration during the interval. The speed control plan for the accel and decel intervals are defined by the following acceleration changing segments s_{accel} and s_{decel}, respectively:

$$\begin{aligned} s_{accel} &= \{I_{accel}, t_f, v_0, a_{max}\}, \\ s_{decel} &= \{I_{decel}, t_f, v_p, -a_{max}\}, \end{aligned}$$

where

$$t_f = \frac{\sqrt{v_0^2 + 2a_{max}|I_{accel}|} - v_0}{a_{max}},$$

which is the time spent in the accel (or decel) interval. The set $\mathcal{J}'$ of location jobs in Case 1-1 and 1-2 is the output of the trimming procedure for $\mathcal{J}$. Sets $\mathcal{J}_1$ and $\mathcal{J}_2$ in Case 2 are subsets of $\mathcal{J}'$ defined as follows:

$$\begin{aligned} \mathcal{J}_1 &= \{\tau | \tau \in \mathcal{J}', I(\tau) \subseteq I_1\}, \\ \mathcal{J}_2 &= \{\tau | \tau \in \mathcal{J}', I(\tau) \subseteq I_2\}. \end{aligned}$$

The time t_s in Case 2 needs a special treatment when $v_p = v_0 = 0$. In this case, the tight interval degenerates to a point, where there are simple location jobs having zero-length

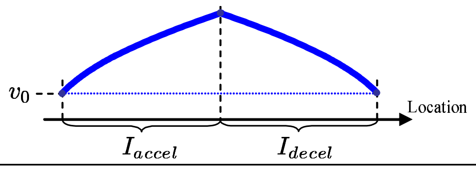
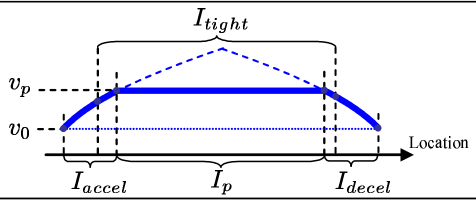
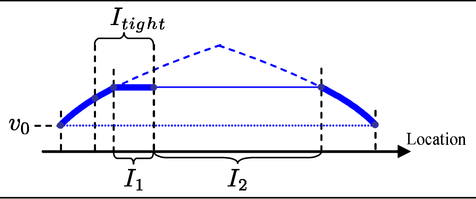
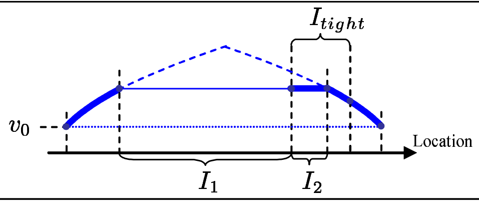
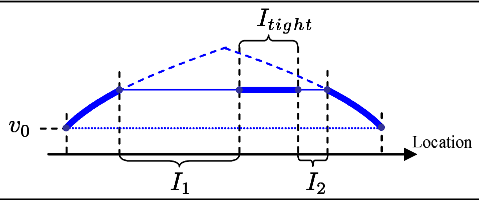
	Location-speed profile	Output
Case 0-1: $I_{tight} = \emptyset$		$\{s_{accel}, s_{decel}\}$
Case 0-2: $I_p \in I_{tight}$	 I_{tight} v_p v_0 Location I_{accel} I_p I_{decel}	$\{s_{accel},$ $\{I_p, \lvert I_p \rvert / v_p, v_p, 0\},$ $s_{decel}\}$
Case 1-1: $I_{accel} \cap I_{tight} \neq \emptyset$ and $I_{decel} \cap I_{tight} = \emptyset$		$\{s_{accel},$ $\{I_1, \lvert I_1 \rvert / v_p, v_p, 0\},$ $\text{MAXIMIZE}(\mathcal{J}', I_2, v_p),$ $s_{decel}\}$
Case 1-2: $I_{accel} \cap I_{tight} = \emptyset$ and $I_{decel} \cap I_{tight} \neq \emptyset$		$\{s_{accel},$ $\text{MAXIMIZE}(\mathcal{J}', I_1, v_p),$ $\{I_2, \lvert I_2 \rvert / v_p, v_p, 0\},$ $s_{decel}\}$
Case 2: $I_{accel} \cap I_{tight} = \emptyset$ and $I_{decel} \cap I_{tight} = \emptyset$		$\{s_{accel},$ $\text{MAXIMIZE}(\mathcal{J}_1, I_1, v_p),$ $\{I_{tight}, t_s, v_p, 0\},$ $\text{MAXIMIZE}(\mathcal{J}_2, I_2, v_p),$ $s_{decel}\}$

Figure 5.11: Possible cases of recursive maximization: the number of recursions differs according to the orientation of the tight interval. Sets of location jobs $\mathcal{J}'$, $\mathcal{J}_1$, and $\mathcal{J}_2$ are determined by the trimming procedure.

feasible location interval. We set t_s as follows so that these location jobs are finished:

$$t_s = \begin{cases} \textsc{Stop-Time}(\mathcal{J}, I_{tight}) & \text{if } v_p = v_0 = 0 \\ |I_{tight}|/v_p & \text{otherwise} \end{cases}$$

where Stop-Time returns the sum of execution time of the location jobs that have zero-length feasible location interval at I_{tight}, which is in fact a single point.

5.4.3 Discussions

Correctness

First we show that the heuristic algorithm always halts. In each recursion, the algorithm identifies a tight interval. Since each tight interval completely contains at least one job, the number of jobs in the next recursion is strictly less than that of the current step. Therefore, the total number of jobs eventually become zero and recursion ends at this point.

Next, we show that the heuristic algorithm produces a feasible speed control plan. We show every procedure in the algorithm preserves the feasibility. One recursion step increases the plateau speed until there is a tight interval. Since the plateau speed is chosen such that there is no interval $[r(\tau), d(\tau')]$ in which processor demand exceeds the time duration allocated to the interval, if the initial speed control plan is feasible, the output is also feasible.

Feasibility preserving property of the trimming procedure is due to the optimality of the EDD and LRT algorithms. For any given feasible speed control plan, the corresponding scheduling problem is schedulable by EDD, because we schedule the simplified set of jobs instead of the original set of general location jobs. When trimming off the accel interval, the EDD algorithm is emulated for the interval and the feasible location intervals of the remaining location jobs are trimmed according to the result. Since the remaining set of location jobs is also schedulable by EDD, the procedure preserves the feasibility. For the decel interval, we do the same thing backwards from the end and emulate the LRT algorithm, which also works backwards. After running EDD for the accel interval and LRT for the decel interval, it is obvious that the remaining set of location jobs for the plateau interval is also feasible. When dividing jobs at the tight interval, assuming $v_p > 0$, EDD is applied to the plateau interval up to the left end of the tight interval. As we discussed, the remaining set of location jobs on the right of the tight interval continues to be feasible. When $v_p = 0$, we can divide the feasible location intervals arbitrarily without violating feasibility, since the plateau speed is still zero.

Robustness

For the heuristic algorithm to be useful in practice, it should be robust against various errors that occur in the real world. Major examples of such errors include unexpected death of nodes and communication failure within communication range.

When a node dies unexpectedly, the data mule can detect it by not receiving any response from the node, assuming that a simple request-response protocol is used. Then the data mule can safely skip the schedule for the dead node and continue data collections from the remaining nodes, without affecting the travel time and data collection performance. Updating the plan at runtime to further improve the travel time is also feasible, but requires more computation at the data mule.

Regarding communication failure, we have assumed that the release locations and the deadline locations of all location jobs are known. However, in reality, they may not be deterministically known due to fluctuations of link quality. One way to deal with this issue is to set these values conservatively. Doing so will usually result in longer travel time, but may be preferred when it is important for the application to gather all data for sure.

5.4.4 Extensions

When endpoint speed is unconstrained

Although our heuristic algorithm assumes the speed at both 0 and L is constrained to zero, we can use it also for the unconstrained case in the following way. The idea is to run the algorithm for a hypothetical travel interval $[X'_s, X'_d]$ that contains the original travel interval $[0, L]$ so that we can freely change the speed at 0 and L.

1. Convert all general location jobs to simple location jobs by SIMPLIFY-JOBSET
2. Identify one tight location interval I_t; let $PD(I_t)$ denote the processor demand for I_t.
3. Calculate the maximum possible speed v_b at one endpoint of I_t. As shown in Figure 5.12, there are two possible cases depending on the processor demand in I_t. When

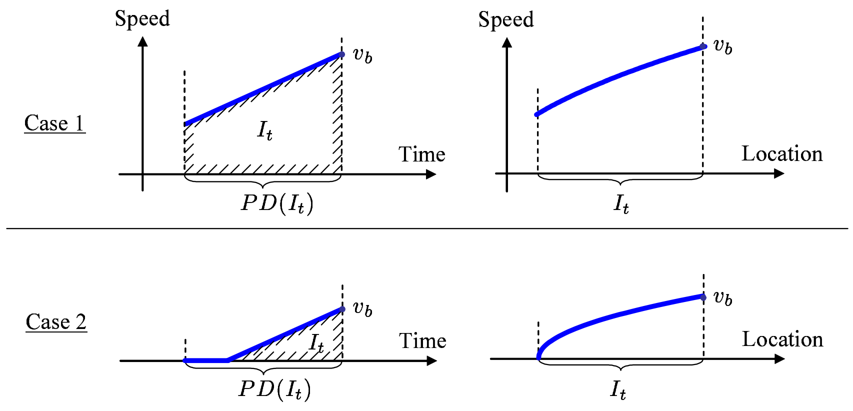


Figure 5.12: Maximum speed at the edge of each location interval as determined by the processor demand

$|I_t|/PD(I_t) - a_{max} \cdot PD(I_t)/2 \geq 0$, Case 1 applies and we obtain

$$v_b = \frac{|I_t|}{PD(I_t)} + \frac{a_{max} \cdot PD(I_t)}{2}. \tag{5.43}$$

Otherwise, Case 2 applies and

$$v_b = \sqrt{2a_{max} \cdot |I_t|}. \tag{5.44}$$

4. Calculate the maximum possible speed at 0 and L. Let v_s and v_d denote the maximum speed at these points. Tentatively assuming there is no other tight interval except I_t, v_s is achieved when the speed at $l[I_t](\equiv x_l)$ is v_b. Similarly v_d is achieved when the speed at $h[I_t](\equiv x_h)$ is v_b. Then we have the equations $v_s^2 - v_b^2 = 2a_{max}x_l$ and $v_d^2 - v_b^2 = 2a_{max}(L - x_h)$. Thus,

$$v_s = \sqrt{v_b^2 + 2a_{max}x_l}, \tag{5.45}$$

$$v_d = \sqrt{v_b^2 + 2a_{max}(L - x_h)}. \tag{5.46}$$

5. Calculate the hypothetical travel interval $[X_s', X_d']$. We should set X_s' sufficiently far from 0 so that the data mule can take any speed up to v_s at 0. Similarly for X_d'. Then we have $v_s^2 = 2a_{max}(0 - X_s')$ and $v_d^2 = 2a_{max}(X_d' - L)$ and obtain

$$X_s' = -\frac{v_s^2}{2a_{max}}, \tag{5.47}$$

$$X_d' = L + \frac{v_d^2}{2a_{max}}. \tag{5.48}$$

For the hypothetical travel interval $[X'_s, X'_d]$, we run MAXIMIZE and recursively maximize the speed, as in the constrained case. We then trim off the speed changes outside $[0, L]$ from the output. Note all of Equations (5.43) to (5.48) are closed-form solutions, and so we can calculate X'_s and X'_d efficiently once we obtain I_t and $PD(I_t)$.

When the maximum speed is constrained

When the speed is restricted to be in the range $[0, v_{max}]$, we make the following changes to the heuristic algorithm:

- In the maximize step, after Line 4 of MAXIMIZE, if $v_p > v_{max}$, then set $v_p \leftarrow v_{max}$ and $I_{tight} \leftarrow \emptyset$, since there is no tight interval. In addition, define a new location interval I_{max} and set $I_{max} \leftarrow [h[I_{accel}], l[I_{decel}]]$.
- In the trim step, do nothing.
- In the recursion step, make no recursive call and output $\{s_{accel}, s_{max}, s_{decel}\}$, where $s_{max} = \{I_{max}, |I_{max}|/v_{max}, v_{max}, 0\}$.

Periodic data generation case

For the periodic data generation case described in Section 4.5.1, we can estimate the travel time T_t iteratively in the following manner:

- Solve the linear program for the variable speed model by ignoring the acceleration constraint. Set the result to the initial value of $\hat{T}_t$, the estimate of T_t.
- Repeat
  - Run the heuristic algorithm with setting $e(\tau) = \lambda(\tau)(\hat{T}_t + T_b)/R$. Denote the travel time as $\tilde{T}_t$.
  - If $|\tilde{T}_t - \hat{T}_t| < \epsilon$, break the loop. Otherwise, update $\hat{T}_t$ by $\tilde{T}_t$ and repeat.

5.5 Analysis of Lower Bound

To evaluate the goodness of heuristic solutions, we need a reference. Since obtaining the global optimal solution for Generalized 1-D DMS problem is hard, we find a lower bound instead.

We present two different lower bounds. First one is only for simple location jobs and based on the analysis of maximum possible speed at each location. Second one is a more general method that uses a QP (quadratic programming) formulation and SDP (semidefinite programming) relaxation.

5.5.1 Lower Bound for Simple Location Jobs Case

Procedure LB-MaxSpeed finds a lower bound of travel time from the following two conditions:

1. Maximum speed condition at each release or deadline location (denoted x_i afterwards)

2. Minimum time condition for each location interval $[x_i, x_j]$

Note that these are necessary conditions, and thus there may not exist a travel whose travel time equals to the derived lower bound.

The first condition is about the maximum speed at each x_i. By the processor demand analysis similar to the one in Section 5.4.2, for each location interval consisting of the release location of a job and the deadline location of another job, the maximum possible speed at the edge of the interval is determined. By extrapolating the speed change within the interval, we obtain an upper bound on the speed at every x_i on the whole travel interval. The details of the procedure is described by the following pseudocode:

```
1: for all location interval $I = [r(\tau), d(\tau')]$ s.t. $\tau, \tau' \in \mathcal{J}, r(\tau) \le d(\tau')$ do
2:     if $|I|/PD(I) - a_{max} \cdot PD(I)/2 > 0$ then
3:         $v_b \leftarrow |I|/PD(I) + a_{max} \cdot PD(I)/2$
4:     else
5:         $v_b \leftarrow \sqrt{2a_{max} \cdot |I|}$
6:     end if
7:     for all $i$ do
8:         if $x_i \notin I$ then
9:             $d_1 \leftarrow \min\{|l[I] - x_i|, |h[I] - x_i|\}$
10:            $v_i[I] \leftarrow \sqrt{v_b^2 + 2a_{max}d_1}$                ▷ Case 1
11:        else
12:            if $|I|/PD(I) - a_{max} \cdot PD(I)/2 > 0$ then
```

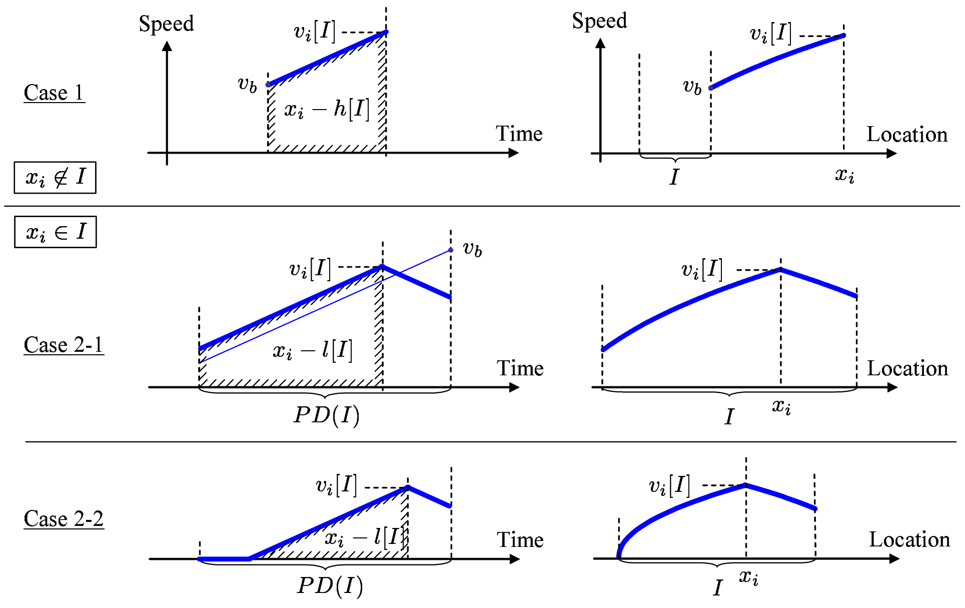


Figure 5.13: Upper bound of the maximum speed at each x_i

```
13:                 $v_i[I] \leftarrow v_b$                       ▷ Case 2-1, loose bound
14:             else
15:                 $d_2 \leftarrow \max\{x_i - l[I], h[I] - x_i\}$
16:                 $v_i[I] \leftarrow \sqrt{2a_{max}d_2}$                 ▷ Case 2-2
17:             end if
18:         end if
19:     end for
20: end for
21: for all $i$ do
22:     $v_i \leftarrow \min_I v_i[I]$
23: end for
```

First, in Lines 2-6, the maximum speed v_b at the edge of the location interval I is calculated in the same way as in Section 5.4.4 (see Figure 5.12). Then in Lines 7-19, we calculate the upper bound $v_i[I]$ of the speed at each x_i, derived from the processor demand constraint of the location interval I. When x_i is out of I (Lines 9-10), $v_i[I]$ is achieved by simply accelerating from the closer edge of I (Figure 5.13, Case 1). It is more complicated when x_i is within I (Lines 12-17). There are two possible cases, as shown in the bottom half of Figure 5.13. In Case 2-1, $v_i[I]$ is derived as the solution of a cubic equation and very complicated. Instead, since we can show $v_i[I] \leq v_b$ is always satisfied when $x_i \in I$, we use v_b as a loose upper bound (Line 13). In Case 2-2, we can calculate $v_i[I]$ exactly (Line 16). Finally, in Line 22, the maximum speed at each x_i is determined by the lowest of all the upper bounds.

The second condition is about the minimum time duration for each location interval $[x_i, x_j]$, which is determined by the processor demand for the interval. By using variables z_i to denote the time to stay in the interval $[x_i, x_{i+1}]$, the condition is represented as a set of linear constraints. Then, the maximum speed at each x_i derived from the first condition (v_i) gives additional constraints for z_i. Combining these constraints, we have the following linear program and obtain a lower bound of travel time by solving this:

Minimize $\sum_i z_i$

Subject to

- (Fastest travel)

$$z_i \geq \frac{2}{a_{max}} \sqrt{a_{max}(x_{i+1} - x_i) + \frac{v_i^2 + v_{i+1}^2}{2}} - \frac{v_i + v_{i+1}}{a_{max}}$$

  The right hand side is the minimum possible travel time starting from x_i at the speed v_i and finishing at x_{i+1} at the speed v_{i+1}. This is a linear constraint since there is no variable (i.e., z_i) on the right hand side.

- (Processor demand) For each location interval $I = [r(\tau), d(\tau')]$ s.t. $\tau, \tau' \in \mathcal{J}, r(\tau) \leq d(\tau')$,

$$\sum_{\tau_L \in \mathcal{J}, I(\tau) \subseteq I} e(\tau) \leq \sum_{[x_i, x_{i+1}] \subseteq I} z_i.$$

5.5.2 Lower Bound based on Quadratic Programming

We present another way of obtaining a lower bound by first formulating the problem as a QP (quadratic program) and then using SDP (semidefinite programming) relaxation to efficiently find a lower bound. Different from the first method, this one is applicable to any problems including general location jobs case.

QP Formulation

Figure 5.14 shows the relationship between time and speed under constrained acceleration. Suppose the data mule moves from location x_i to x_{i+1}. Also suppose that

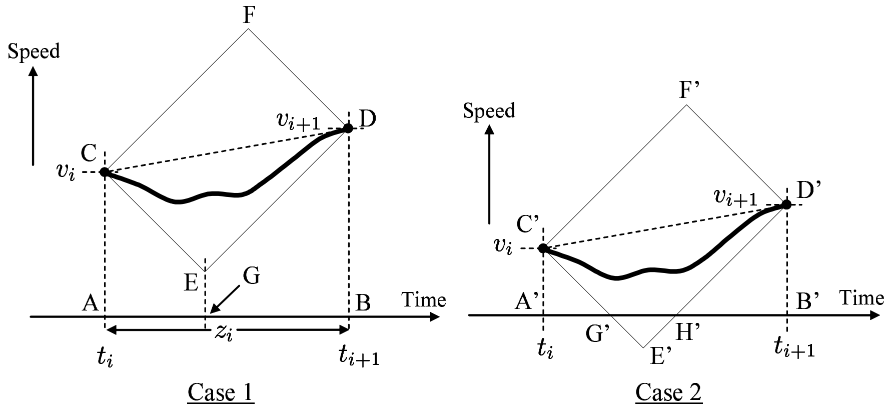


Figure 5.14: Time and speed under constrained acceleration: All possible transitions from point C (t_i, v_i) to D (t_{i+1}, v_{i+1}) are confined in the parallelogram CFDE in Case (a) or the pentagon C'F'D'H'G' in Case 2.

time is t_i and the speed is v_i at location x_i. As the figure shows, there are two different cases.

We focus on Case 1 first. Any change of speed over time is expressed as a curve in the time-speed graph. Under the constraint on maximum absolute acceleration, all possible changes of speed during the time interval $[t_i, t_{i+1}]$ are confined in the parallelogram CFDE, where the slope is a_{max} for CF and ED, and $-a_{max}$ for FD and CE. As the area between the curve and the time axis corresponds to the distance traveled, we have the following relationships:

$$x_{i+1} - x_i \geq S(ACEDB), \tag{5.49}$$

$$x_{i+1} - x_i \leq S(ACFDB), \tag{5.50}$$

where $S(\cdot)$ is the area. These areas are calculated as follows:

$$S(ACEDB) = \alpha - \beta, \tag{5.51}$$

$$S(ACFDB) = \alpha + \beta, \tag{5.52}$$

where

$$\alpha = \frac{1}{2}(v_i + v_{i+1})z_i, \tag{5.53}$$

$$\beta = \frac{1}{4}\left\{a_{max}z_i^2 - \frac{(v_{i+1} - v_i)^2}{a_{max}}\right\}, \tag{5.54}$$

$$z_i = t_{i+1} - t_i. \tag{5.55}$$

As for Case 2 in Figure 5.14, since we assume the movement is one-way, the speed must be nonnegative at any time. Thus the change of speed along C'-E'-D' is impossible. Instead, the speed change along C'-G'-H'-D' achieves the shortest travel distance, which is equal to $S(A'C'G') + S(H'D'B')$. In this case, the following constraint replaces (5.49) in Case 1:

$$\begin{aligned} x_{i+1} - x_i &\geq S(A'C'G') + S(H'D'B') \\ &= \frac{v_i^2 + v_{i+1}^2}{2a_{max}}. \end{aligned} \tag{5.56}$$

Now we combine these two cases. Since the vertical coordinate of point E (or E') is $(v_i + v_{i+1} - a_{max}z_i)/2$, we can summarize the constraints as follows:

$$x_{i+1} - x_i \geq \alpha - \beta \tag{5.57}$$

$$x_{i+1} - x_i \leq \alpha + \beta \tag{5.58}$$

$$x_{i+1} - x_i \geq \frac{v_i^2 + v_{i+1}^2}{2a_{max}} \quad (\text{if } v_i + v_{i+1} - a_{max}z_i \leq 0) \tag{5.59}$$

Note (5.57) can be a constraint for both cases, since (5.59) is a stronger constraint than (5.57) for Case 2.

Now, in addition to v_i and z_i, we define a variable $p_i(\tau)$ to represent the time allocated to location job τ in the location interval $[x_i, x_{i+1}]$. Then we obtain the following quadratic programming problem:

Minimize $\sum_i z_i$

Subject to

- (One-way movement) $z_i \geq 0, v_i \geq 0$
- (Job completion) $\sum_i p_i(\tau) = e(\tau)$
- (Feasible interval) $p_i(\tau) = 0$ if $\forall I \in \mathcal{I}(\tau), [x_i, x_{i+1}] \not\subseteq I$
- (Processor demand) $\sum_{\tau \in \mathcal{J}} p_i(\tau) \leq z_i$
- (Maximum absolute acceleration)

$$\alpha - \beta \leq x_{i+1} - x_i \leq \alpha + \beta \tag{5.60}$$

$$\frac{v_i^2 + v_{i+1}^2}{2a_{max}} \leq x_{i+1} - x_i \quad (\text{if } v_i + v_{i+1} - a_{max}z_i \leq 0) \tag{5.61}$$

$$|v_{i+1} - v_i| \leq a_{max}z_i \tag{5.62}$$

where α and β are defined in Equations (5.53) and (5.54), respectively.

The constraint (5.61) is not quadratic since it is conditioned, but it is replaced by equivalent quadratic constraints using additional variables b_i and w_i as follows:

$$\begin{aligned}
b_i(1-b_i) &= 0\\
b_i(v_i+v_{i+1}-a_{max}z_i) &\geq 0\\
(1-b_i)(v_i+v_{i+1}-a_{max}z_i) &\leq 0\\
(1-b_i)(w_i-2a_{max}(x_{i+1}-x_i)) &\leq 0\\
v_{i+1}^2+v_i^2 &= w_i
\end{aligned}$$

SDP (semidefinite programming) Relaxation

We construct a relaxation problem for the nonconvex QP problem above. The domain of a relaxation problem contains that of the original problem, so the minimum value of the relaxation problem gives a lower bound of the original problem, assuming it is a minimization problem. We first omit the constraint (5.61) for simplification[8] and then apply SDP relaxation. SDP is a convex optimization problem and can thus be solved efficiently (see [BV04] etc.).

In our formulation, (5.60) is the only quadratic constraint after we omit (5.61). We introduce a matrix variable $X \equiv xx^T$ where $x = [v^T|z^T]^T$ and replace (5.60) by the following linear constraint:

$$\alpha' - \beta' \;\leq\; x_{i+1} - x_i \;\leq\; \alpha' + \beta', \tag{5.63}$$

where

$$\begin{aligned}
\alpha' &= \frac{1}{2}(X_{i,n+i}+X_{i+1,n+i}),\\
\beta' &= \frac{1}{4}\left\{a_{max}X_{n+i,n+i} - \frac{(X_{i+1,i+1}-2X_{i,i+1}+X_{i,i})}{a_{max}}\right\}.
\end{aligned}$$

SDP relaxation replaces the equality constraint $X = xx^T$ by $X - xx^T \succeq 0$, where the symbol "$\succeq$" denotes componentwise inequality. By using the Schur complement [BV04, §A.5.5], we convert it to a positive semidefiniteness constraint:

$$\begin{bmatrix} X & x \\ x^T & 1 \end{bmatrix} \succeq 0 \tag{5.64}$$

[8]SDP relaxation without omitting (5.61) is possible, but the size of the problem becomes large and numerical computation is very hard.

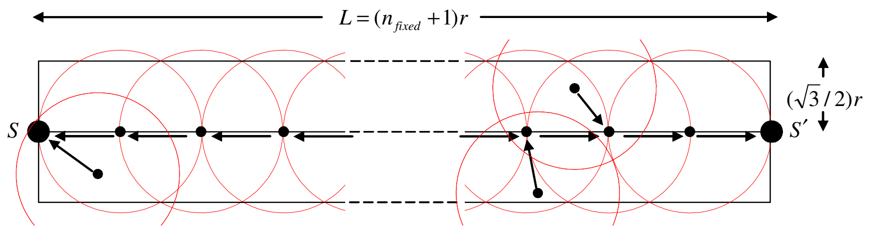


Figure 5.15: Randomly generating test cases: Fixed nodes are aligned on line SS'; (b) For multiple data mules case. An example with two data mules is shown.

The resulting problem is an SDP problem, since it only has linear constraints and one positive semidefiniteness constraint.

5.6 Performance Evaluation

In this section, we evaluate the performance of the heuristic algorithm by numerical experiments.

5.6.1 Comparison with Multihop Forwarding

For the first experiment, we compare the data delivery latency and energy consumption between data mule approach and multihop forwarding approach.

Method

We use MATLAB for simulation experiments. Besides the heuristic algorithms, we have implemented a naive method for speed control. In the naive method, a data mule stops to collect data from each node one by one. Specifically, a data mule moves to the point on the path that is the closest to the node, collects data from the node while stopping, and moves to the next point. This is similar to the one used in [MY07], though they do not assume an acceleration constraint.

We randomly generate test cases in the following ways for each of the two experiments. For the first experiment, we need to guarantee the connectivity between the base station and each node, since otherwise multihop forwarding approach is not feasible. For this purpose, we generate node placements as shown in Figure 5.15, in which we arrange n_{fixed} "fixed" nodes on the horizontal dotted line SS', each of them are r apart. Base

Table 5.1: Comparison with multihop forwarding: Amount of transmission is relative to the minimum possible amount.

	Data mule		Multihop forwarding
	Heuristic	Naive	
Latency (sec)	502.1	1098.3	(251.0)
Amount of transmission (max/avg)	1.0/1.0	1.0/1.0	25.1/6.0

stations are located at S and S', assuming the data mule starts from and comes back to the base station. Then we put $n - n_{fixed}$ "random" nodes at random locations in the rectangle within the distance $(\sqrt{3}/2)r$ from the horizontal line. In this way we can guarantee that each of these random nodes has at least one fixed node or the base station within its communication range and thus is reachable to the base station via multihop forwarding. In case of data mule approach, the data mule moves from S to S' on the straight line. All the nodes are within r from the trajectory and thus the data mule can collect data from them. We set $n_{fixed} = 20$, $n = 50$, and $r = 100[m]$ in the experiment.

We set the execution time $e = 10[sec]$. Each experiment is repeated for 10 times on each case and the average is used as the result. We use the variable speed with acceleration constraint as the mobility model of data mules and set $a_{max} = 1[m/s^2]$ and $v_{max} = 10[m/s]$ to roughly simulate the mobility capability of a radio controlled helicopter used in [TMF+07].

Results

We compare data mule approach and multihop forwarding approach in terms of latency and energy consumption. We use one data mule in both of the heuristic and the naive algorithms for data mule approach. For a multihop forwarding algorithm, we use a simple one that each node forwards the data that it generated and received from other nodes to its neighbor node that is closest to the base station.

Table 5.1 shows the latency and energy consumption in these approaches averaged for 10 different node placements. Latency is measured by the total travel time in case of data mule approach. For multihop forwarding approach, we use a lower bound calculated from the maximum amount of data that a single node needs to send. Latency is more than twice in data mule approach compared to the multihop forwarding algorithm[9].

[9]It should be noted that the latency in multihop forwarding will be shorter in periodic data generation case, since the nodes can keep streaming the data.

Within data mule approach, latency in the heuristic algorithm was less than half of that in the naive method.

Amount of transmission is minimum in data mule approach, since each node sends its own data to the data mule and does not forward other nodes' data. On the other hand, it is 6.0 on average and 25.1 at maximum in the multihop forwarding algorithm. Since communications account for major part of energy consumption at nodes, these results suggest that data mule approach is more energy efficient roughly by six times. Moreover, since the maximum amount of transmission occurs at the node next to the base station and reachability to the base station is lost without this node, functional lifetime in multihop forwarding is much shorter in this simple forwarding algorithm compared to data mule approach.

5.6.2 Comparison with Lower Bounds

Method

We implemented the heuristic algorithm, LB-MaxSpeed method, and the SDP relaxation of the QP formulation. We developed and ran the programs on MATLAB 7.4.0 (R2007a) with YALMIP interface [Löf04] and used SeDuMi [Stu99] for SDP solver.

We randomly generate test cases in the following way. In Figure 5.16, the horizontal dotted line in the middle of the rectangle represents the path of data mule. L is the total travel length. We randomly place circles of radius r so that each circle has its center inside the rectangle. A circle represents the communication range of a sensor node and the center is the location of the node. In this way, we can make the circle intersect with the path of data mule. This intersection corresponds to a feasible location interval. The length of each feasible location interval is between 0 and $2r$. We place nk circles, where n is the number of location jobs and k is the number of feasible location intervals per job.

For the experiments, we set $r = 10[m]$ to roughly simulate the communication range of IEEE 802.15.4. We define "length factor" f and set $L = fn$ so that we can keep the density of nodes unchanged for different n's. We set the execution time for each job to $10[sec]$. Each experiment is repeated for 100 times on each case and the average and standard deviation are calculated.

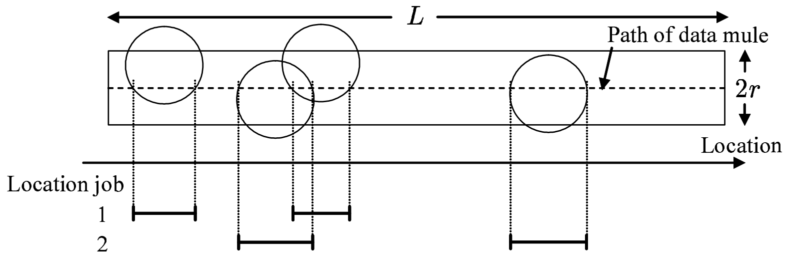


Figure 5.16: Randomly generating test cases: two location jobs; each has two feasible location intervals

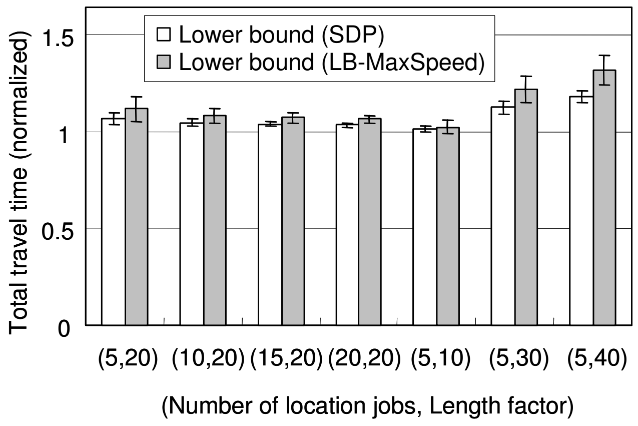


Figure 5.17: Comparison of lower bounds

Results and Discussions

To evaluate the results, we use the following two metrics: quality and scalability. Quality is measured by the total travel time and scalability is measured by the elapsed time to compute the schedule.

Since we evaluate the heuristic algorithm by comparing with the lower bounds, a lower bound should be as tight as possible for the evaluation to be more accurate. In Figure 5.17, we compare the lower bounds from LB-MaxSpeed method with the ones from the SDP relaxation for several different cases. From the figure, we can observe LB-MaxSpeed constantly yields higher (thus tighter) lower bound for all cases. Thus, for simple location jobs, we compare LB-MaxSpeed with the heuristic algorithm. We use the SDP relaxation for general location jobs case, as LB-MaxSpeed is only applicable for simple location jobs.

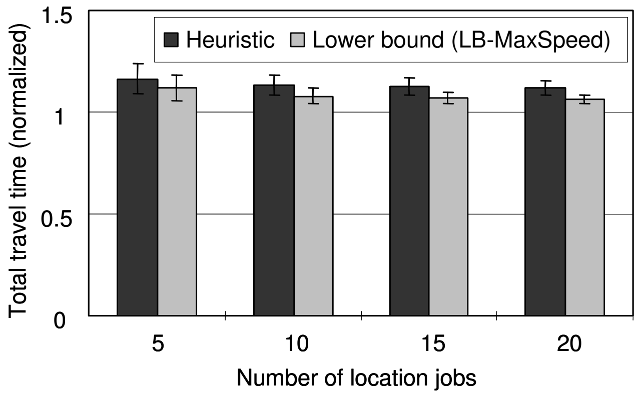


Figure 5.18: Effect of number of location jobs

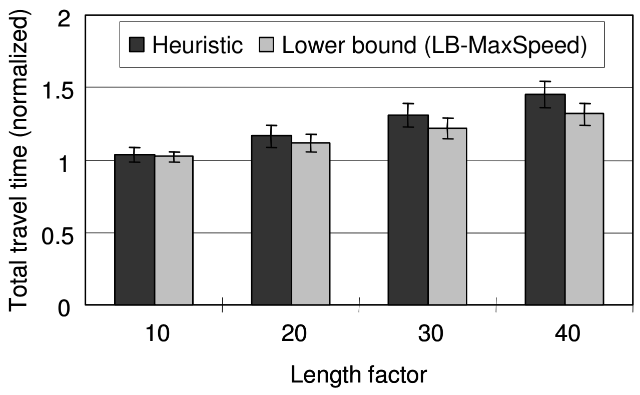


Figure 5.19: Effect of density: node density is less for larger length factor

Quality To compare the total travel time, we first normalize the travel time by the sum of execution time of all jobs, which serves as the trivial lower bound. By this normalization, we can roughly estimate the relative quality of solution of the heuristic algorithm for different test cases.

Figure 5.18 shows the effect of varying number of location jobs from $n = 5$ to 20. We use $k = 1$ (simple location jobs) and length factor $f = 20$. For this range of n, the schedules from the heuristic algorithm have up to 5% (on average) longer total travel time than those from LB-MaxSpeed. The average ratio slightly increases (1.042 for $n = 5$, 1.052 for $n = 20$), but it does not imply the heuristic algorithm performs poorer for larger n, since we do not know how close the lower bound is to the optimal solution. What we can say for sure is that the global optimal solution lies between the

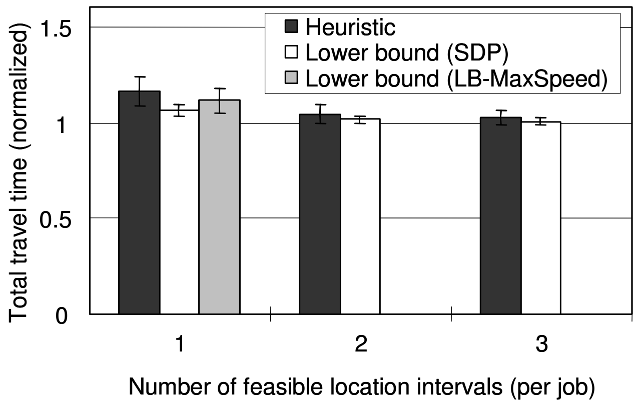


Figure 5.20: Effect of number of feasible location intervals

lower bound and the heuristic solution. Therefore, for these test cases, the total travel time of the heuristic algorithm is within 5% (on average) of the global optimal solution.

Figure 5.19 shows the effect of varying density by changing the length factor from $f = 10$ to 40. Smaller length factor means higher node density. We use the number of location jobs $n = 5$ and $k = 1$ (simple location jobs). The average ratio to the lower bound varies from 1.010 ($f = 10$) to 1.103 ($f = 40$), but again we cannot conclude the heuristic algorithm performs poorer for less node density.

Figure 5.20 shows the effect of varying number of feasible location intervals from $k = 1$ to 3. We use the number of location jobs $n = 5$ and $f = 20$. For the cases of $k = 2$ and 3, we use the lower bound from the SDP relaxation. When $k = 1$, the average ratio to the lower bound from LB-MaxSpeed is 1.042 and that for the SDP relaxation is 1.094. As k increases, the ratio decreased (1.029 for $k = 2$, 1.018 for $k = 3$, both to the SDP relaxation).

All these results suggest that the heuristic algorithm yields reasonably good schedules within 10% (on average) of the optimal solutions for these test cases.

Scalability Figure 5.21 shows the computation time of the heuristic algorithm for the problems of different size. We use $k = 1$ (simple location jobs) and length factor $f = 20$, and varied the number of location jobs from $n = 5$ to $n = 200$. We have ran our code on MATLAB (HW: Intel Pentium 4 2.8GHz (RAM:1.5GB), OS: Linux (kernel 2.6.20)). On average, the heuristic algorithm takes 0.8 second for 100 location jobs and 3.2 seconds for 200 location jobs. It is considerably fast, given the SDP relaxation takes more than

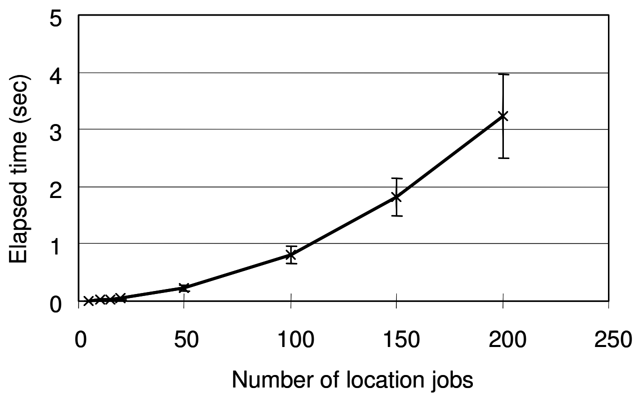


Figure 5.21: Computation time

20 minutes on the same machine to compute the lower bound for $n = 20$ case (data not shown).

5.7 Connections with Speed Scaling Problems

As discussed in Section 4.6, we can see the speed $v(t)$ of data mule corresponds to the inverse of the processor speed $s(t)$ in a special case of speed scaling problem, in which the power function is given as $P(s) = s$. The generalized mobility model with acceleration constraint roughly corresponds to the speed scaling problem with a constraint on the rate of processor speed change. In this section we first define the rate-constrained speed scaling problem and show how we can apply the approximation scheme for the Generalized 1-D DMS problem to it.

5.7.1 Rate-constrained Speed Scaling Problem

We define the rate-constrained speed scaling problem by adding a constant K, which is the maximum rate of processor speed change, to the input and the rate constraint $|ds(t)/dt| \leq K$. The assumption of constrained speed change rate has been validated through experiments [NYM97], but most studies on speed scaling adopt simple settings without the rate constraint.

We can apply the approximation scheme for the Generalized 1-D DMS problem to the rate-constrained speed scaling problems. For that purpose, we first define "cumulative processed workload," which is a function of t and represents the total workload processed until t. This corresponds to the cumulative execution time that we have used in the 1-D DMS problem.

A major difference between the 1-D DMS problem and speed scaling problems is in the objective function. The 1-D DMS problem roughly corresponds to a special case of speed scaling problems where $P(s) = s$. Nevertheless, as discussed below, we can apply the same technique to more general rate-constrained speed scaling problems, since we can determine the speed function that minimizes the energy consumption, as long as $P(s)$ is positive and monotonically increasing for s.

Another difference is that we cannot apply the analysis on short intervals to the rate-constrained speed scaling problem. In the Generalized 1-D DMS problem, we have allowed a speed-quantized trajectory to take more time in a short interval than the original trajectory does. However, this is not possible for the rate-constrained speed scaling problem because the dominant points in the speed scaling problem are defined on the time axis and taking more time means exceeding the dominant point.

In the rest of the section, we present the necessary changes in the approximation scheme to apply that to the rate-constrained speed scaling problem.

5.7.2 Energy-minimizing Speed Function

For a given feasible schedule $\mathcal{S} = (s(t), job(t))$, there are infinitely many different speed functions that can replace $s(t)$ without violating the feasibility. Since the total energy cost varies depending on the speed function, we need to determine the one that minimizes the cost.

In the following theorem, we show that the 3-phase speed changes we used in Lemma 5.3.7 are the energy-minimizing speed function.

Theorem 5.7.1. *Given time t_0, t_1, the processor speed s_0, s_1 at each time, and workload W, let $s^*(t)$ denote a function that minimizes energy consumption $\int_{t_0}^{t_1} P(s(t))\,dt$ while satisfying $\int_{t_0}^{t_1} s(t)dt = W$ where $P(s)$ is positive and monotonically increasing. Then, $s^*(t)$ is defined as follows:*

Case (I): if $W_2 \le W \le W_{max}$,

$$s^*(t) = \begin{cases} s_0 + K(t - t_0) & \text{if } t_0 \le t \le t_a \\ s_p & \text{if } t_a \le t \le t_b \\ s_1 + K(t_1 - t) & \text{if } t_b \le t \le t_1 \end{cases} \tag{5.65}$$

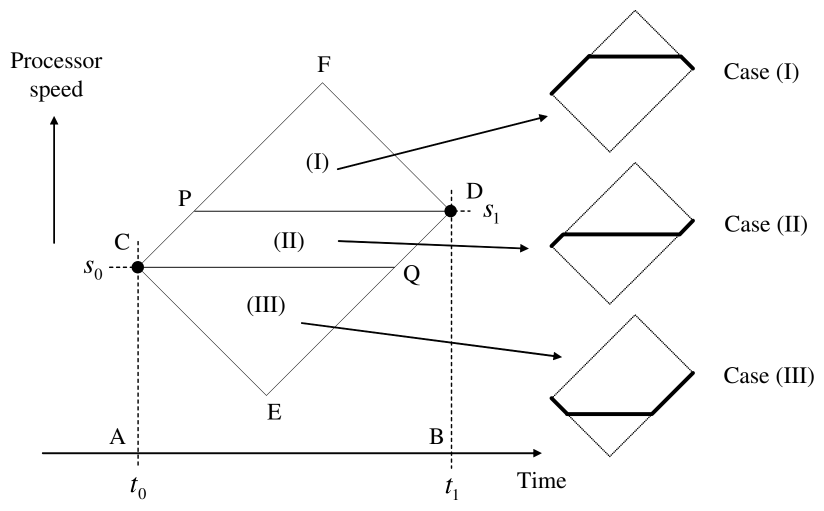


Figure 5.22: Shape of the energy-minimizing speed function

Case (II): if $W_1 \le W \le W_2$*,*

$$s^*(t) = \begin{cases} s_0 + K(t - t_0) & \text{if } t_0 \le t \le t_a \\ s_p & \text{if } t_a \le t \le t_b \\ s_1 - K(t_1 - t) & \text{if } t_b \le t \le t_1 \end{cases} \tag{5.66}$$

Case (III): if $W_{min} \le W \le W_1$*,*

$$s^*(t) = \begin{cases} s_0 - K(t - t_0) & \text{if } t_0 \le t \le t_a \\ s_p & \text{if } t_a \le t \le t_b \\ s_1 - K(t_1 - t) & \text{if } t_b \le t \le t_1 \end{cases} \tag{5.67}$$

where

$$W_{min} = S(ACEDB), \qquad W_{max} = S(ACFDB)$$
$$W_1 = S(ACQDB), \qquad W_2 = S(ACPDB)$$

($S(\cdot)$ is the area of shape) in Figure 5.22, t_a, t_b are the time of acceleration changes determined by W, and s_p is the speed during $[t_a, t_b]$.

Proof. Suppose $s(t)$ is another speed function that satisfies $\int_{t_0}^{t_1} s(t)dt = W$ and not identical to $s^*(t)$. We show that total energy consumption in $s(t)$ is larger than that in $s^*(t)$.

Let D denote the region bounded by $s(t)$ and $s^*(t)$. Let $l(s)$ denote the total length of intersection of D and a horizontal line at s. Further, we define two disjoint

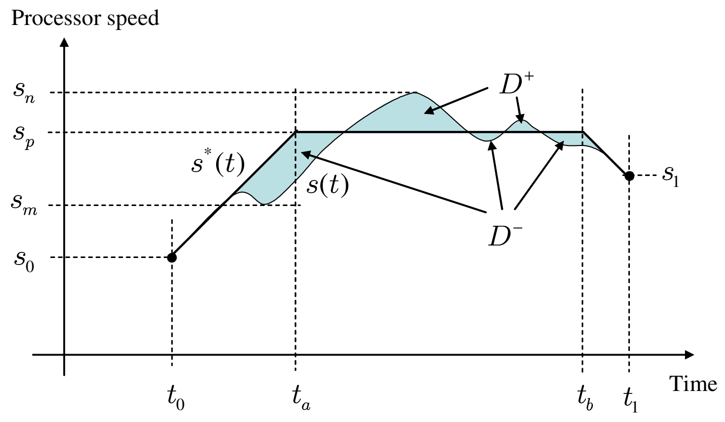


Figure 5.23: Energy-minimizing speed function: $s(t)$ and $s^*(t)$ process the same workload in $[t_0, t_1]$, but $s(t)$ consumes more energy than $s^*(t)$.

subregions of D as follows: D^+ is the subregion of D above $s^*(t)$ and D^- is the rest. Since the processed workload is identical in $s^*(t)$ and $s(t)$, we have $S(D^+) = S(D^-)$, where $S(\cdot)$ represents the area of a region.

Figure 5.23 shows an illustration for Case (I). Since $s^*(t)$ accelerates at the maximum rate in $[t_0, t_a]$, $s(t)$ cannot exceed $s^*(t)$. Similarly, $s(t) \leq s^*(t)$ in $[t_b, t_1]$. Thus, region D^+ is located only above the line $s = s_p$. Clearly, region D^- is located only below the line. These can be easily shown for Case (II) and (III) in a similar way. Then we have

$$S(D^+) = \int_{s_p}^{s_n} l(s)ds, \qquad S(D^-) = \int_{s_m}^{s_p} l(s)ds,$$

where s_m, s_n are the minimum and maximum of $s(t)$ in $[t_0, t_1]$, respectively.

Let E_Δ denote the difference of energy consumption between $s^*(t)$ and $s(t)$. Since $P(s)$ is positive and monotonically increasing, we obtain

$$\begin{aligned} E_\Delta &= \int_{t_0}^{t_1} \left(P(s(t)) - P(s^*(t))\right) dt \\ &= \int_{s_p}^{s_n} P(s)l(s)ds - \int_{s_m}^{s_p} P(s)l(s)ds \\ &\geq P(s_p)\left(\int_{s_p}^{s_n} l(s)ds - \int_{s_m}^{s_p} l(s)ds\right) \\ &= 0. \end{aligned}$$

Therefore $s^*(t)$ minimizes the energy consumption. $\square$

5.7.3 Necessary Changes in the Approximation Scheme

Dominant points are now defined on the time axis at each r_j, d_j in addition to 0 and T. The quantization step for processor speed is derived in the same way as in Eq. (5.35) as follows:

$$q_i = \min\left\{\frac{\epsilon}{8}\frac{K}{s_{max}}\min_i\{t_{i+1}-t_i\}, \frac{\epsilon^2 s_{max}}{4}\right\}. \tag{5.68}$$

From a similar argument as in Lemma 5.3.14, the quantization step for cumulative processed workload is $w = \{\epsilon/(2n+1)\}\min_i\{R_i\}$.

The DP algorithm in Section 5.3.5 is modified as follows. The table is indexed by (time, processor speed, cumulative processed workload). The entry in (i, j, k) is the total energy consumption of the energy-minimizing speed function from $(0, 0)$ to (t_i, jq_i) that has cumulative processed workload kw at time t_i. The table is initialized by 0 at $(0, 0, 0)$ and $+\infty$ otherwise. The final output is the entry in $(N_d - 1, 0, \sum R_m)$.

Since there are no type-II transitions, the update rules are as follows. A transition from (i, j, k) to $(i+1, j', k')$ is possible when all of the following constraints are satisfied. We define $s = jq_i$ and $s' = j'q_{i+1}$.

- (Valid speed change) $|s' - s| \le K(t_{i+1} - t_i)$.
- (Feasibility) $\sum_{m:d_m \le t_{i+1}} R_m \le k'w \le \sum_{m:r_m \le t_{i+1}} R_m$. Note that there are no type-II transitions.
- (Cumulative processed workload) $k'w - kw \le L_{max}(K, s_{max}, t_{i+1} - t_i, s, s')$, where the right hand side is the maximum possible workload processed when starting from $(time, speed) = (t_i, s)$ and ending at (t_{i+1}, s'). This is because actual processed workload $(k'w - kw)$ must be less than the maximum possible processed workload. Note that we do not need the lower bound condition, since the processor can be idle.

When all these conditions are met, we can update the entry at $(i + 1, j', k')$. Let C and C' denote the current entry at (i, j, k) and $(i + 1, j', k')$, respectively. We update the entry by $C + \int_{t_i}^{t_{i+1}} P(s^*_W(t))dt$ if this is less than C', where $s^*_W(t)$ is the energy-minimizing speed function when workload is W. Here, W is defined as $W = \max(k'w - kw, L_{min}(K, s_{max}, t_{i+1} - t_i, s, s'))$.

5.8 Related Work

We introduce some related work on speed scaling with rate constraint. Related work in job scheduling is introduced in Chapter 4.

There are a few papers on speed scaling that adopts the assumption of constrained rate of processor speed change. Hong et al. [HQPS98] studied the problem and presented several findings about the relations among processor speed, incurred time delay and workload, which are directly applicable to the data mule scheduling problem. They presented a heuristic algorithm for a limited case, in which the processor speed at each dominant point is given, but did not give a detailed analysis on the general case. Our approximation scheme is built upon their analysis and gives a guaranteed performance for the general case of the problem.

Yuan and Qu [YQ05] classify the models of speed scaling into “ideal”, “multiple”, and “feasible”. “Ideal” allows continuous voltage levels and “multiple” only allows discrete levels. “Feasible” allows continuous levels but the maximum voltage change rate is constrained. It is further classified into “optimistic feasible” and “pessimistic feasible”. A prominent difference between these two models is whether a task can be processed during transition to the new voltage level: it is allowed in “optimistic feasible” model and not in “pessimistic feasible” model. In this classification, our work and Hong et al.’s work [HQPS98] are classified as the “optimistic feasible” model.

5.9 Summary

In this chapter we have studied the 1-D DMS problem with the generalized mobility model that has an acceleration constraint. We have shown NP-hardness for general location jobs case and designed an approximation scheme that runs in time polynomial to the job parameters. We have also designed an efficient heuristic algorithm. Through the analysis of lower bound and numerical experiments, we have demonstrated that the heuristic algorithm runs fast and finds near-optimal solutions that are within 10% of the lower bound. Finally we have discussed the similarity with the rate-constrained speed scaling problem and shown how we can apply the approximation scheme to this problem.

Acknowledgements This chapter, in part, has been accepted for publication as “Optimal Speed Control of Mobile Node for Data Collection in Sensor Networks” by Ryo

Sugihara and Rajesh K. Gupta in IEEE Transactions on Mobile Computing [SG09b]. The dissertation author was the primary investigator and author of this paper.

This chapter, in part, has been submitted for publication as “Complexity of Motion Planning of Data Mule for Data Collection in Wireless Sensor Networks” by Ryo Sugihara and Rajesh K. Gupta in Theoretical Computer Science [SG09a]. The dissertation author was the primary investigator and author of this paper.

Chapter 6

Path Selection

In this chapter we discuss the path selection problem. The main idea of the DMS problem framework is to divide the problem into loosely connected subproblems. Accordingly, path selection is treated as an independent problem. We present one possible formulation as a graph problem, which we call Label-Covering Tour problem. We first define the problem and determine the cost metric through preliminary experiments. Since the Label-Covering Tour is NP-hard, we present an approximation algorithm. Further we evaluate the performance by simulation experiments.

6.1 Label-Covering Tour Problem

6.1.1 Idea of Formulation

The ultimate objective of the path selection problem is to find a path such that the shortest travel time can be realized in the corresponding 1-D DMS problem. However, it is not clear which path results in shorter travel time. For example, even if the path length is short, the travel time would be long if the intersections of the path and communication range of each node are short, because the data mule needs to slow down to collect all the data. Moreover, it is also difficult to search an optimal path in a brute-force manner when the data mule can freely move around within the space.

To deal with these issues, we simplify the path selection problem. To reduce the solution space, we consider a complete graph having vertices at sensor nodes' locations and assume the data mule moves between vertices along a straight line. Each edge is associated with a cost and a set of labels, where the latter represents the set of nodes

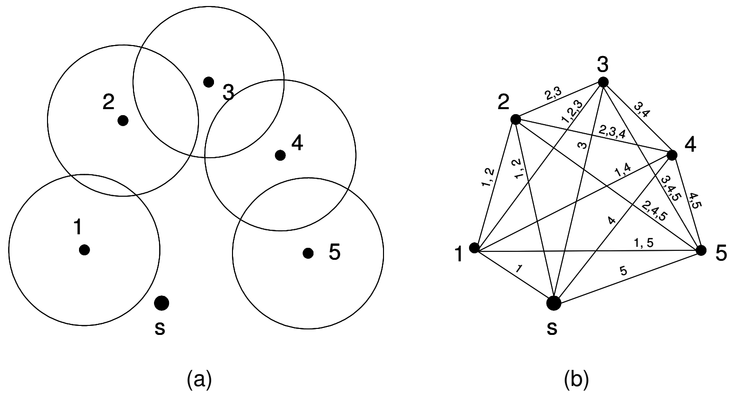


Figure 6.1: Simplifying the path selection problem using a labeled graph representation: (a) Instance of path selection problem. (b) Corresponding labeled graph.

whose communication ranges intersect with this edge. In other words, the data mule can collect data from these nodes while traveling along this edge. We want to find a minimum-cost tour that the data mule can collect data from all the nodes. We discuss later how we assign the cost to each edge so that a tour with smaller cost results in shorter travel time.

Figure 6.1 is an example that depicts the basic idea of the formulation. Figure 6.1(a) shows five nodes and their communication ranges, in addition to the base station (shown as "s"), where the data mule starts and brings the data back. From this input, we construct a labeled undirected complete graph as shown in Figure 6.1(b). Each edge e has a set of labels $L(e) \subseteq L$ and cost $c(e)$, where $L = \{l_1, ..., l_n\}$ is the set of all labels and n is the number of sensor nodes. We determine $L(e)$ as follows: $l_i \in L(e)$ if node i's communication range intersects edge e. Intuitively, by moving along edge e, the data mule can collect data from the nodes whose labels are in $L(e)$.

6.1.2 Problem Definition

Now we define the Label-Covering Tour problem formally as follows: Given an undirected complete graph $G = (V, E)$ where each vertex in $V = \{v_0, v_1, ..., v_n\}$ is a point in $\mathbf{R}^2$, a cost function on edges $c : E \rightarrow \mathbf{Q_0^+}$, a set $L = \{l_1, ..., l_n\}$ of labels, and a constant r. Each edge $e_{ij} \in E$ is associated with subset $L_{ij} \subseteq L$. For $k = 1, ..., n$, $l_k \in L_{ij}$ iff the Euclidean distance between v_k and an edge e_{ij} is equal to or less than r. A tour T is a list of points that starts and ends with v_0, allowing multiple visits to each point. A tour T is "label-covering" when it satisfies at least one of the followings for $k = 1, ..., n$: 1) $\exists e_{ij} \in T(E), l_k \in L_{ij}$, where $T(E)$ is the set of edges traversed by T, or 2) $dist(v_0, v_k) \leq r$, where $dist(v_i, v_j)$ is the Euclidean distance between v_i and v_j. Find a label-covering tour T that minimizes the total cost $\sum_{e_{ij} \in T(E)} c_{ij}$.

The choice of cost metric is discussed in the next section.

Unfortunately, this simplified problem is still NP-hard.

Theorem 6.1.1. *Label-Covering Tour is NP-hard.*

Proof. We show metric TSP is a special case of Label-Covering Tour. First we choose the cost function c to satisfy the triangle inequality (e.g., Euclidean distance). For a given set of points $V = \{v_0, ..., v_n\}$, by choosing a small r, we can make $dist(v_0, v_i) > r$ for all $i > 0$, $L_{ij} = \{l_i, l_j\}$ for all $i, j > 0$, and $L_{0j} = \{l_j\}$ for all $j > 0$. For such r, any label-covering tour must visit all the points. An optimal label-covering tour does not visit any point multiple times except x_0 at the start and the end of the tour, since in such cases, we can construct another label-covering tour with smaller total cost by "shortcutting." Therefore, an optimal label-covering tour is an optimal TSP tour for V. □

6.1.3 Other Formulations of Path Selection

The Label-Covering Tour problem is merely one possible way of formulating the path selection problem. Clearly TSP is also an option, as used in many studies, though it has a problem that we cannot benefit from the communication range. An alternative formulation that takes account the communication range is TSP with Neighborhoods (TSPN) problem, in which the problem is to find a shortest tour that visits each of given regions. Application of TSPN problem to path selection is proposed in some recent studies [YOS07, TLTI08]. Although the solutions are better in TSPN than in Label-Covering Tour in terms of the total Euclidean distance, we adopt the latter for the following reasons:

- Tours tend to touch the communication ranges at the boundary in TSPN. Even if we assume circular communication range, wireless link quality at the boundary is worse than near the center in reality. This would result in longer time for transmitting data and thus it is likely that the benefit in travel time is not as much as that in path length, when compared with the tours from the Label-Covering Tour problem.

- The number of vertices in TSPN tours are generally close to the number of regions. This makes the data mule stop often, which leads to loss of time when it takes non-negligible time to change direction or there is an acceleration constraint. On the other hand, in Label-Covering Tour problem, the number of vertices is smaller when the communication range is larger.

- Some approximation algorithms of TSPN may produce tours with arcs (e.g., [DM03]), which may be non-preferable due to difficulty of maneuvering. Most approximation algorithms are based on constructing TSP tours connecting representing points, each of which is chosen from each region (e.g., [DM03, EFMS05, YOS07]). This is modeled by slightly extending the definition of Label-Covering Tour, specifically by putting vertices at other locations as well as the node locations, labeling the additional edges accordingly, and finding a label-covering tour for that graph.

6.2 Choice of Cost Metric

In the definition of Label-Covering Tour, the cost c_{ij} is a critical parameter. In a restricted scenario, in which the data mule can either move at a constant speed or stop and no remote communication is used, Euclidean distance is the optimal cost metric in the sense that the shortest travel time is realized when the total path length is minimum. However, it is not clear for the general case in which the speed is variable under an acceleration constraint.

Since we minimize the total cost, and also we want to choose a tour that we can achieve the shortest travel time, a good cost metric should be strongly correlated with the travel time. Therefore we measure the goodness of cost metric by the correlation coefficient between cost and total travel time in the corresponding 1-D DMS problem. When the correlation is high, smaller cost implies shorter total travel time, and thus finding a minimum cost tour makes more sense.

We compare three different cost metrics that seem reasonable:

- Number of edges: $c_{ij} = 1$

- Euclidean distance: $c_{ij} = dist(x_i, x_j)$

- Uncovered distance: $c_{ij} = \sum_{s \subseteq e_{ij}, \forall k, dist(x_k, s) > r} |s|$, i.e., total length of intervals in edge e_{ij} that are not within the communication ranges of any nodes.

Uncovered distance is apparently a reasonable cost metric because it represents the total distance that the data mule "wastes", i.e., travels without communicating with any nodes.

6.2.1 Experimental Methods

We assume nodes are deployed in the circular area of radius d that has a start (i.e., point x_0) in the center. We randomly place other nodes within the circle so that they are uniformly distributed. For each edge connecting a pair of nodes, we assign a set of labels by calculating the distance from the line segment and each node.

For each of the node deployments, we randomly generate label-covering tours. A tour is generated by random walk which, at each point, chooses next point randomly, repeats this until all the labels are covered, and goes back to x_0. We measure the cost of the tour in three different cost metrics as listed above.

Using the tour, we transform the original problem to 1-D DMS problem and find a near-optimal latency by using the heuristic algorithm presented in Section 5.4. We assume that each node has the same execution time e and the communication range r, and also that the speed of data mule needs to be zero at each point where it changes the direction[1]. For each node deployment, the cost and the total travel time are normalized among different random tours so that the mean is zero and standard deviation is one. For the collective set of data for same d, r, and e, we calculate the correlation coefficient between the normalized cost and the normalized total travel time for each cost metric.

For each (d, r, e), we generate 1000 examples, which consist of 20 random tours for each of 50 node deployments. We use $d = 150, 500$ and $r = 10, 100$ for Label-Covering Tour. For 1-D DMS, we use the execution time $e = 2, 20$, the maximum absolute acceleration $a_{max} = 1$, and the maximum speed $v_{max} = 10$. These parameter values roughly simulate data collection by a helicopter as in [TMF$^+$07] by assigning units as follows: meters for d and r, seconds for e, m/s^2 for a_{max}, and m/s for v_{max}. Also the values of r are chosen to simulate the communication ranges of IEEE 802.15.4 and 802.11, respectively.

[1]Otherwise, it would require infinite acceleration, since we assume a path consists of only line segments and not curves.

Table 6.1: Correlation coefficients between total cost and total travel time for different cost metrics: 20 nodes, $a_{max} = 1$, $v_{max} = 10$.

Radius (d)	150				500			
Comm. range (r)	10		100		10		100	
Exec. time (e)	2	20	2	20	2	20	2	20
Num. edge	0.992	0.987	0.982	0.850	0.984	0.982	0.988	0.988
Euclidean dist.	0.997	0.996	0.990	0.835	0.999	0.999	0.999	0.999
Uncovered dist.	0.992	0.993	—	—	0.999	0.999	0.935	0.935

6.2.2 Results

Table 6.1 shows the correlation coefficients. Except one case, both number of edges and Euclidean distance had correlation coefficient above 0.98, and Euclidean distance had higher correlation than number of edges. In the exceptional case $(d, r, e) = (150, 100, 20)$, the correlation was weaker than other cases. This is most likely because the travel time is more influenced by the execution time rather than the moving time, since the deployment area is small relative to the size of communication range and also the execution time is long. Uncovered distance had similar results but had no data when $(d, r, e) = (150, 100, 2)$ and $(d, r, e) = (150, 100, 20)$. These are the cases when the communication range is so large that the total cost measured by uncovered distance is always zero.

These results suggest that number of edges and Euclidean distance are both appropriate metrics that precisely measure the goodness of paths. In the rest of the paper, we use Euclidean distance as the cost metric.

6.3 Approximation Algorithm

We design an approximation algorithm for the Label-Covering Tour problem. As discussed earlier, Euclidean distance is used as the cost metric. This enables us to design an approximation algorithm by using known algorithms for metric TSP where the triangle inequality holds. Figure 6.2 shows the approximation algorithm for Label-Covering Tour. It first finds a TSP tour T by using any algorithm (exact or approximate) for TSP. Then, using dynamic programming, it finds a short label-covering tour that is obtained by shortcutting T. For the dynamic programming, we use two tables $d[i]$ and $tour[i]$. Letting $T(i)$ denote the i-th vertex in tour T, $tour[i]$ is the shortest of all the paths from $T(0)$ to $T(i)$ that are obtained by shortcutting $T(0)T(1)T(2)...T(i)$ and cover

```
• Make a TSP tour $T$ using an exact or approximation algorithm for metric TSP
• Initialize $d[0] \leftarrow 0$, $d[1...n] \leftarrow +\infty$, $tour[0] \leftarrow \{T(0)\}$, and $tour[1...n] \leftarrow \emptyset$.
• For $i = 0$ to $n-1$ do
    – For $j = i+1$ to $n$ do
        ∗ Check if the line segment $T(i)T(j)$ is within distance $r$ from each of the nodes $T(i+1), ..., T(j-1)$.
        ∗ If yes and $d[i] + |T(i)T(j)| < d[j]$, update the tables by $d[j] \leftarrow d[i] + |T(i)T(j)|$ and $tour[j] \leftarrow append(tour[i], T(j))$.
• Return $tour[n]$.
```

Figure 6.2: Approximation algorithm for Label-Covering Tour: $T(i)$ is the i-th vertex that the tour T visits. $T(0)$ is the starting vertex.

the labels $T(0), ..., T(i)$. $d[i]$ is the length of $tour[i]$.

Computation time of the algorithm is $\mathcal{C}_{TSP} + O(n^3)$, where $\mathcal{C}_{TSP}$ denotes the computation time of the algorithm used for solving TSP.

Next we analyze the approximation factor of the algorithm. Let T_{OPT}, T_{APP} denote the optimal label-covering tour and the approximate label-covering tour, respectively. Total length of tour T is denoted as $|T|$. Also let α be the approximation factor of the TSP algorithm used in the first step of the approximation algorithm. Then we have the following theorem:

Theorem 6.3.1. $|T_{APP}| \leq \alpha(|T_{OPT}| + 2nr)$

Proof. Clearly $|T_{APP}| \leq \alpha|T_{TSP}|$, where T_{TSP} is the optimal TSP tour. We give a lower bound to T_{OPT} by constructing another TSP tour by modifying T_{OPT}. Figure 6.3 shows the idea of construction. The points A and B (shown in filled circles) are visited by T_{OPT} and other points in the figure (shown in non-filled circles) are not. We call the former "visited points" and the latter "non-visited points". By the definition of a label-covering tour, any non-visited points are within distance r from either a traversed edge or a visited point of a label-covering tour. For example in the figure, all of AC and DD', ..., GG' have the length less than r. Then we can construct a "tour"[2] that is

[2]This is not a tour in our definition because it does not consist of edges between the nodes.

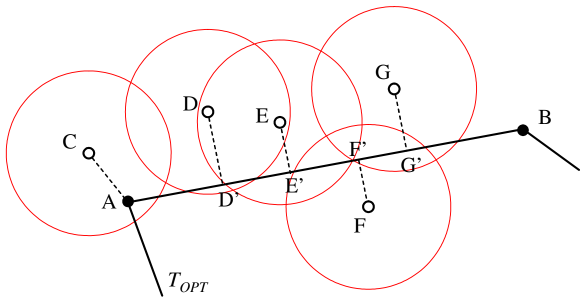


Figure 6.3: Constructing a TSP tour from the optimal label-covering tour T_{OPT}: every non-visited point is within distance r from T_{OPT}.

identical to T_{OPT} but takes a detour to visit each non-visited point (e.g., ACAD'DD'...B). Since there are at most n non-visited points, total length of detour is at most $2nr$. This "tour" is easily converted to a shorter TSP tour by skipping all additional points (e.g., D', E', ...) and apply shortcutting so that each point is visited exactly once. Therefore, we have $|T_{OPT}| + 2nr \geq |T_{TSP}|$. The theorem follows by combining this and $|T_{APP}| \leq \alpha |T_{TSP}|$. $\square$

6.4 Performance Evaluation

We evaluate the performance of the approximation algorithm by numerical experiments. We have implemented the algorithm in MATLAB.

6.4.1 Method

We deploy n nodes in the circular area of radius d that has the base station at the center. The nodes are randomly placed within the circle. Each of the nodes has circular communication range of radius r. For each edge connecting a pair of nodes, we assign a set of labels by calculating the distance from the line segment and each node. We use Concorde TSP solver [Con] to find an optimal TSP tour.

Using the tour, we transform the original problem to 1-D DMS problem and calculate the travel time by using the heuristic algorithm presented in Section 5.4. We assume that each node has the same execution time $e = 10[sec]$ and also that the speed of data mule needs to be zero at each point where it changes the direction[3].

[3]Otherwise, it would require infinite acceleration, since we assume a path consists of only line segments and not curves.

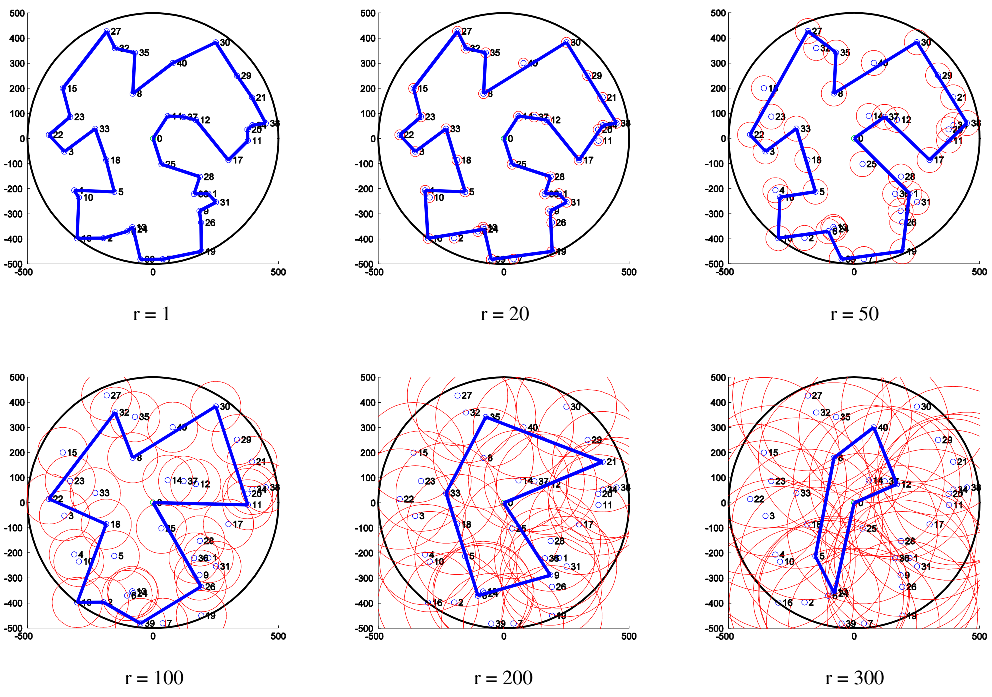


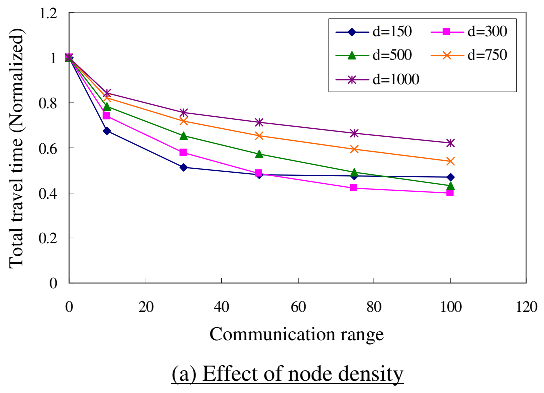
Figure 6.4: Label-covering tours for different communication ranges: 40 nodes, $d = 500$; Path of data mule is shown in bold line.

For each (n, d, r), we generate 50 node deployments and take the average for the results. For 1-D DMS, we use the maximum absolute acceleration $a_{max} = 1[m/s^2]$, and the maximum speed $v_{max} = 10[m/s]$ to roughly simulate the motion of a helicopter as in [MFL+08].

Figure 6.4 shows some examples of label-covering tours for a node deployment with different communication ranges. As the communication range grows, the number of visited points becomes less and the path length becomes shorter.

6.4.2 Effect of Node Density and Network Size

Figure 6.5(a) shows the relation between the communication range and the total travel time for different node density. To see how the size of communication range affects the travel time, we have normalized the total travel time by the one when the communication range is zero. The graph shows that the total travel time is reduced in all cases by up to 60% for this parameter set, suggesting the proposed problem formulation and algorithm altogether successfully exploit the breadth of communication range. The



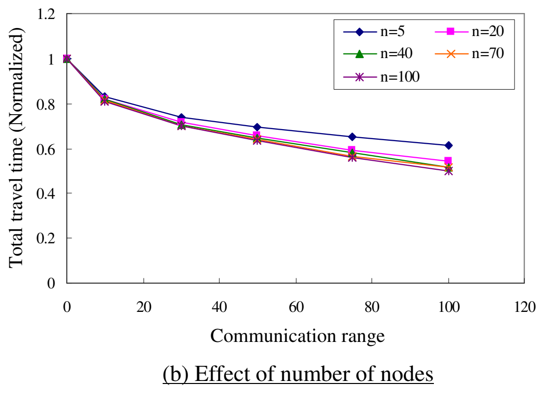


Figure 6.5: Comparison of total travel time for (a) different node density (40 nodes) and (b) different number of nodes ($d = 500$ for 20 nodes): $a_{max} = 1$, $v_{max} = 10$.

amount of reduction is bigger when the density is higher (i.e., smaller d), except the case of $d = 150$ for large communication ranges. This is because the total travel time is already very close to the lower bound, which is the product of the execution time and the number of nodes.

Figure 6.5(b) shows the effect of number of nodes, varied from $n = 5$ to $n = 100$. We set d to 500 when $n = 20$, and changed d in proportion to $\sqrt{n}$ so that the density remains constant. The results show the reduction of total travel time for large communication ranges, but no big difference for different number of nodes.

6.4.3 Comparison with Other Strategies

Next we compare the travel time of our approximation algorithm with those of other algorithms as listed below.

- TSP-like: Based on the model used in [SRS04]. Data mule visits all nodes. It stops at each node location to collect data and moves to the next node. While moving, the speed is constant at v_{max}. We use optimal TSP tours.

- Stop-and-collect: Based on the model used in [MY07]. Data mule takes a label-covering tour, as in our approximation algorithm. However, it stops to collect data when it is in the communication range of each node. While moving, speed is constant at v_{max}. We find tours by using our approximation algorithm with optimal TSP tours[4].

[4]We could not use the path selection algorithm proposed in [MY07], since it has a restriction

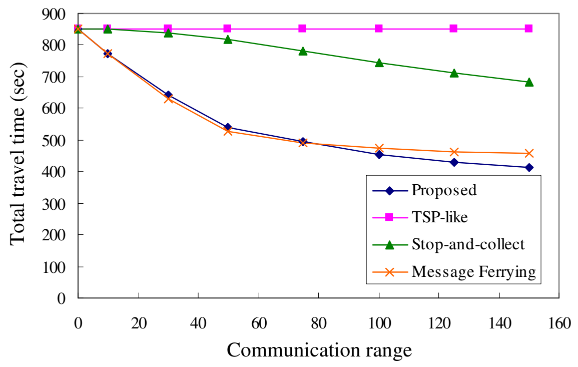


Figure 6.6: Comparison of total travel time for different path selection algorithms: 40 nodes, $d = 500$, $a_{max} = +\infty$, $v_{max} = 10$.

- Message Ferrying: Based on the algorithm proposed in [ZA03]. Data mule visits all nodes as in the TSP-like algorithm, but communication is also done while moving. Speed is variable between 0 and v_{max}. Speed and data collection schedule are determined by solving a linear program such that the total travel time is minimized. We use optimal TSP tours.

To allow direct comparison, we set $a_{max} = +\infty$ for our proposed approximation algorithm, since all other algorithms assume data mule can change its speed instantly. Note that when $a_{max} = +\infty$, we can obtain an exact solution for the 1-D DMS problem by solving a linear program, as discussed in Section 4.4.

Figure 6.6 shows the results for a representative case for 40 nodes. When the communication range is small, the travel time does not differ among the algorithms. As the communication range grows, Message Ferrying and the proposed algorithm show larger improvements than other two methods, and the proposed algorithm gets gradually better than Message Ferrying. When the communication range is 150, the proposed algorithm is nearly 10% better than Message Ferrying, 40% better than Stop-and-collect, and more than 50% better than TSP-like method.

When there is an acceleration constraint (i.e., $a_{max} \neq +\infty$), which none of these studies has addressed, the gaps between the proposed algorithm and others are expected to be larger. This is because all of these methods require the data mule to stop more

on the configuration of data mule and deployment area. Specifically, it assumes the data mule starts from the left end of the deployment area, travels toward the right end, and comes back to the initial position.

frequently than the proposed algorithm does.

These results suggest that the proposed algorithm effectively exploits broader communication range for planning the path of the data mule.

6.5 Related Work

As far as we know there is no previous study on the Label-Covering Tour problem, but there are some studies on similar problems. The Steiner tree problem [GJ79] is similar in choosing a subset of vertices in the graph to minimize the cost metric, but it finds a tree instead of a tour. The Prize Collecting Traveling Salesman Problem (PCTSP) [Bal89] is a variation of the traveling salesman problem and finds a tour of subset of vertices. The difference is that, in PCTSP each vertex v_i has a profit p_i and the problem is to find a minimum length tour such that the total profit of the tour is more than or equal to some value p.

The Covering Tour Problem (CTP) [GLS97] is one of the most related problems and defined as follows.

> Let $G = (V \cup W, E)$ be an undirected graph, where $V \cup W$ is the vertex set, $V = \{v_1, ..., v_n\}$, and $E = \{(v_i, v_j) | v_i, v_j \in V \cup W, i > j\}$ is the edge set. Vertex v_1 is a depot, V is a set of vertices that can be visited, $T \subseteq V$ is a set of vertices that must be visited ($v_1 \in T$), and W is a set of vertices that must be covered. A distance metric $C = (c_{ij})$ satisfying the triangle inequality is defined on E. The CTP consists of determining a minimum length tour or Hamiltonian cycle over a subset of V in such a way that the tour contains all vertices T, and every vertex of W is covered by the tour, i.e., it lies within a distance c from a vertex of the tour.

Gendreau et al. [GLS97] developed an exact branch-and-cut algorithm based on an integer linear program formulation of the CTP problem, and also a heuristic algorithm. There are some differences between the CTP and the Label-Covering Tour problem. For example, there is no distinction between V and W in the Label-Covering Tour problem. In addition, the term "cover" in CTP means the covering by a vertex, as can be seen from that every vertex of W must be within a distance c from a vertex of the tour. On the other hand, in Label-Covering Tour we allow a vertex to be covered not only by a vertex but also by an edge, considering the distance between a point and a line segment.

The traveling salesman problem with neighborhoods (TSPN) is also a closely related problem and is indeed used for formulating the path selection problem. In Section

6.1.3 we have already discussed the pros and cons of using the TSPN problem and why we chose the Label-Covering Tour problem.

6.6 Summary

In this chapter we have presented a formulation of path selection problem as a graph problem called the Label-Covering Tour problem. Since the Label-Covering Tour problem is NP-hard, we have designed an approximation algorithm. We have experimentally demonstrated that the approximation algorithm finds near-optimal paths and achieves much smaller latency compared to previously proposed algorithms, especially when the communication range is large.

Acknowledgements This chapter, in part, has been published as “Improving the Data Delivery Latency in Sensor Networks with Controlled Mobility” by Ryo Sugihara and Rajesh K. Gupta in the proceedings of the 4th IEEE International Conference on Distributed Computing in Sensor Systems (DCOSS) [SG08a]. The dissertation author was the primary investigator and author of this paper.

This chapter, in part, has been submitted for publication as “Path Planning of Data Mules in Sensor Networks” by Ryo Sugihara and Rajesh K. Gupta in ACM Transactions on Sensor Networks [SG09d]. The dissertation author was the primary investigator and author of this paper.

Chapter 7

Hybrid Approach with Multihop Forwarding

In this chapter we consider the combined approach of data mule and multihop forwarding. In the "pure" data mule approach that we have discussed in Chapter 4 – 6, the energy consumption at each node is minimum and the data delivery latency is relatively large. On the other hand, multihop forwarding requires greater energy due to increased data transfer at each node but the latency is expected to be much shorter. In the hybrid approach, these two approaches are combined in such a way that the designers of sensor networks can balance the energy consumption and the data delivery latency according to application needs. We formulate the problem by extending the DMS problem and design centralized and distributed algorithms. Then we implement the combined approach on the ns2 network simulator [ns2] to experimentally evaluate the effectiveness of the formulation and algorithms.

7.1 Forwarding Problem

We consider a combined approach of data mule and multihop forwarding. In the DMS problem framework, we realize this by defining a new "forwarding" problem that is placed in front of the path selection problem as shown in Figure 3.2. The forwarding problem is to determine how much data each node forwards to other nodes and to the data mule while satisfying a predetermined energy consumption constraint.

7.1.1 Problem Description

The objective of the forwarding problem is to find a forwarding plan such that the induced DMS problem has the shortest total travel time. Different from the "pure" data mule approach, in which each node sends its data only to the data mule, it can now forward its data to other neighboring nodes as well. More importantly, if a node decides to forward all data to other nodes, the data mule does not need to collect data directly from this node. Then the data mule can possibly take a shorter path to reduce the travel time.

We present a centralized algorithm based on linear program formulation. Since finding the optimal forwarding plan that minimizes the travel time in the induced DMS problem is at least as hard as the DMS problem, we make it an independent problem by changing the objective function.

We minimize the average distance of nodes from the base station weighted by the amount of data at each node after forwarding. There are three reasons why this is a reasonable choice as the objective function. First, this function is likely to shorten the path of the data mule by forcing the nodes at the edge of network to primarily use forwarding. Secondly, this function allows a smooth transition between the data mule approach and multihop forwarding. As the energy consumption limit grows, more data is forwarded closer to the base station. In a connected network, all the data is eventually forwarded to the base station without using a data mule, which is equivalent to "pure" multihop forwarding. Finally, since the function is linear, we can formulate the problem as a linear program as described below.

7.2 Centralized Algorithm by Linear Programming

We assume the location of sensor nodes and the connectivity between them are known. We also assume the following parameters are given:

- λ_i: Data generation rate of node i
- E_{limit}: Energy consumption limit at each node per unit time
- E_r, E_s: Energy consumption for receiving and sending unit data
- R: Bandwidth, i.e., maximum data rate that each node can communicate with other nodes and the data mule

We use variable x_{ij} to represent the amount of data sent from node i to j per unit time. Let λ'_i denote the data rate that node i sends directly to the data mule. λ'_i is defined by the difference of incoming data rate and outgoing data rate as follows:

$$\lambda'_i = \sum_j x_{ji} + \lambda_i - \sum_j x_{ij}. \tag{7.1}$$

Then we have the following linear program:

Minimize $\sum_i d_i \lambda'_i$

Subject to

- $\forall i$, $x_{ii} = 0$.
- (Connectivity) $\forall i, j$ s.t. $i \neq j$, $x_{ij} \geq 0$ if node j is in the communication range of node i. Otherwise $x_{ij} = 0$.
- (Flow conservation) $\forall i$, $\lambda'_i \geq 0$.
- (Energy consumption) $\forall i$, $E_r \sum_j x_{ji} + E_s \left(\sum_j x_{ji} + \lambda_i \right) \leq E_{limit}$,
- (Bandwidth) $\forall i$, $2 \sum_j x_{ji} + \lambda_i \leq R$.

In the objective function, d_i is the distance between node i and the base station. In the energy consumption constraint, the first term in the left hand side is the amount of energy consumed by receiving data and the second term is that for sending data. About the latter, node i sends $\sum_j x_{ij}$ to other nodes and λ'_i to the data mule per unit time, when averaged over time. Using Eq. (7.1), the sum of these two equals $\sum_j x_{ji} + \lambda_i$. The bandwidth constraint is obtained using the amount of incoming data $\sum_j x_{ji}$ and the outgoing data $\sum_j x_{ij} + \lambda'_i$.

The formulation above is also capable of expressing the case in which each node communicates along the preconstructed routing tree as in [XWXJ07]. This is possible by replacing the connectivity constraint with the following one:

- (Routing tree) $\forall i, j$ s.t. $i \neq j$, $x_{ij} \geq 0$, if node j is node i's parent in the routing tree. Otherwise $x_{ij} = 0$.

7.3 Distributed Algorithm

The LP formulation above yields the optimal forwarding plan in the sense that it minimizes the weighted distance of data from the base station. However in practice, it may be difficult to tell each node about the list of forwarding destination and the data rate for each. To cope with this issue, we present a distributed algorithm where each node determines the forwarding destination and the data rate in a distributed manner.

In the algorithm, we consider the case that each node forwards the data along a routing tree. For connected networks, there is only one routing tree rooted at the base station. For disconnected networks, there are multiple routing trees, one for each connected cluster. In each cluster, the node closest to the base station is chosen as the root. We describe how to identify connected clusters, construct intra-cluster routing trees, and plan the forwarding rate.

7.3.1 Clustering and Constructing Routing Trees

We can simultaneously find connected clusters and construct intra-cluster routing trees by extending DSDV [PB94], which is a routing scheme based on the distance vector algorithm. We extend DSDV so that each node exchanges ID and position of the interim root node. An interim root node is the node that is reachable and closest to the base station as far as the current node knows. The information on root node is updated when the current node knows the one closer to the base station, and is propagated to neighbors when it is updated. These communications can be piggybacked on the update packets of normal DSDV. When it reaches a convergence, each node has the correct information on the root node and the next hop for reaching the root.

To plan the forwarding rate, each node needs to learn the set of immediate child nodes as well as the parent. This is realized by each node sending a message to the parent node along the established routing tree.

7.3.2 Planning the Forwarding Rate

Forwarding rate is calculated in the following three phases.

Request

The request phase is initiated from the leaf nodes and proceeds toward the root. Each node tells the parent the cumulative data rate, which is the total data rate generated at the node and its descendants. Let Λ_i denote cumulative data rate of node i, which is defined as follows:

$$\Lambda_i \equiv \lambda_i + \sum_{j \in C_i} \Lambda_j, \tag{7.2}$$

where C_i is the set of immediate children of node i.

Allocate

The allocate phase proceeds downwards from the root. Parent node tells each immediate child the allocated data rate, which is the maximum data rate that the parent can receive from this child.

Let $y_i^{(in)}$ denote the total data rate that node i receives from its children. Then $y_i^{(in)}$ needs to satisfy

$$E_r y_i^{(in)} + (\lambda_i + y_i^{(in)}) E_s \leq E_{limit}, \tag{7.3}$$

and thus

$$y_i^{(in)} \leq \frac{E_{limit} - \lambda_i E_s}{E_r + E_s}. \tag{7.4}$$

For each child node, we distribute the maximum data rate proportionally by the cumulative data rate. Thus the maximum data rate X_j that child node j can send to the parent i is

$$X_j = \frac{\Lambda_j}{\sum_{k \in C_i} \Lambda_k} \frac{E_{limit} - \lambda_i E_s}{E_r + E_s}. \tag{7.5}$$

Plan

The plan phase proceeds upwards from the leaf nodes. Node determines the forwarding rate and tell it to the parent. For node i, the total data rate $y_i^{(out)}$ to be sent to either the parent or the data mule is

$$y_i^{(out)} = \lambda_i + y_i^{(in)}, \tag{7.6}$$

where $y_i^{(in)} = \sum_{j \in C_i} x_{ji}$.

We try to forward the data to parent node as much as possible and send the remaining data to the data mule. Therefore, if we let node j be the parent of i, data rate x_{ij} is

$$x_{ij} = \min \left\{ y_i^{(out)}, X_i \right\}. \tag{7.7}$$

By setting x_{ij} in this way, inequality (7.4) is satisfied. Data rate λ'_i to the data mule is

$$\lambda'_i = y_i^{(out)} - x_{ij}. \tag{7.8}$$

7.4 Implementing Hybrid Communication Model

In the combined approach of data mule and multihop forwarding, each node potentially communicates with multiple destinations including the data mule. In addition, for the communication with the data mule, each node needs to send the data when the data mule is within the communication range. In this section, we discuss how we implement this hybrid communication model. Figure 7.1 shows the basic idea. The protocol works on top of MAC layer and can work with any MAC protocols.

7.4.1 Node-to-Node Communication

In node-to-node communication for multihop forwarding, each node sends the data to one of the forwarding destinations. To satisfy the energy consumption constraint, it needs to follow the forwarding rate for each of the destinations that is determined by solving the forwarding problem. It also needs to have a storage for the data to be sent to the data mule.

Handling Incoming Data Packets

A node receives data packets from other nodes by node-to-node forwarding. It also generates data by itself periodically at the built-in sensor module. When a node receives these data packets, it adds them at the end of the storage queue.

Choosing a Forwarding Destination

When there is a data packet in the storage, the node may forward it. It sends a packet to one of the forwarding destinations if there is one whose actual forwarding rate is lower than the predetermined rate. It chooses the destination that is most behind the schedule. For this purpose, each node records the cumulative amount of data sent to each destination.

When a node forwards data, it fetches a data packet from the end of the storage queue, which is the packet that is received most recently. This is to avoid the delay to

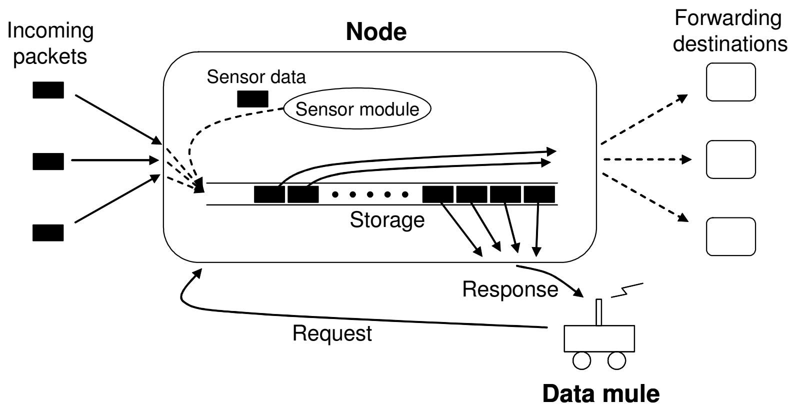


Figure 7.1: Combined data mule and forwarding communications

accumulate during node-to-node forwarding, since it would cause large delivery latency in case a packet is forwarded node-to-node many times.

7.4.2 Communication with the Data Mule

According to the job schedule determined by solving the DMS problem, each node needs to send certain amount of data to the data mule at certain time duration. We enable this by letting the data mule initiate the communications. According to the schedule, the data mule sends a request packet to the node that it collects data from. A request packet contains the information about the size of data to be sent and the request ID. The request ID is unique for each allocation in the job schedule.

When a node receives a request packet from the data mule, it fetches the specified amount of data from the beginning of the storage queue (i.e., received least recently) and sends it back to the data mule packet by packet. The request ID is included in each of the response packets and it serves as an acknowledgment for the request.

When the data mule does not receive any response packets for a request after a certain time period, it sends the same request again.

7.5 Simulation Experiments

We experimentally evaluate the combined approach of data mule and multihop forwarding in the periodic data generation case, specifically on the effectiveness of formulation and algorithms in optimizing the energy-latency trade-off.

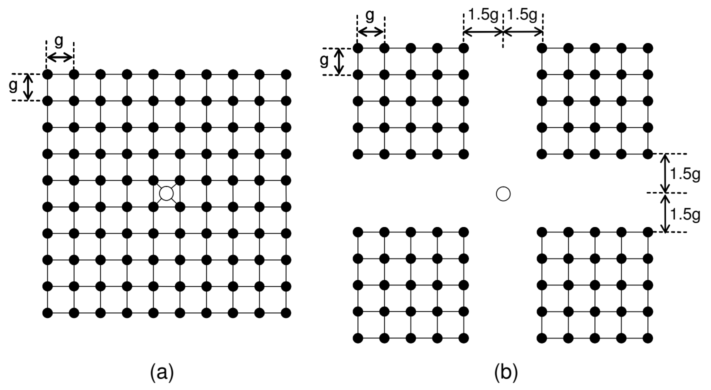


Figure 7.2: Network topology: (a) Connected network; (b) Disconnected network. White circle is the base station. Line between two circles represents that they are within the communication range. Grid size g is set to $0.8r$, where r is the radius of communication range, and a uniformly random disturbance of $[-0.025r, 0.025r]$ is added to the position of each node.

7.5.1 Methods

We have implemented the centralized and distributed algorithms for the forwarding problem and the algorithms for the DMS problem in MATLAB with YALMIP interface [Löf04] and SeDuMi [Stu99] for LP solver. The MATLAB program generates a Tcl script for ns2 [ns2], which simulates the movement of the data mule and the communication among the data mule and the nodes.

To assess the performance, we measure the delivery latency for each data packet from the time it is generated to the time the base station receives it either from neighboring nodes or via the data mule. For each test case, the simulation on ns2 is repeated multiple periods until it reaches stability. We consider it stable when the average delivery latency of the data received in the current period is within $\pm 1\%$ of that of the previous period. If it is stable, we use the data for the next period as the final results.

Figure 7.2 shows two network topologies we use for the experiments. Both of them have 100 sensor nodes, one base station and one data mule, but one is a connected network and the other is a disconnected network. The disconnected network consists of four connected networks of 25 nodes and the base station is not directly reachable from any nodes.

For the data mule's movement, we use the variable speed model. The range of speed is $0 \leq v \leq 10m/s$, which roughly simulates the movement of a UAV used in

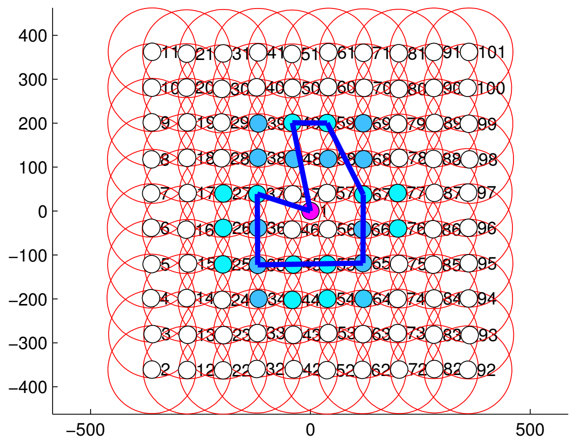


Figure 7.3: Example of forwarding plan and calculated path: Connected network, $E_{limit} = 10E$, centralized forwarding algorithm. Nodes in white forward all data and the data mule does not collect data directly from them. Path is shown in bold line.

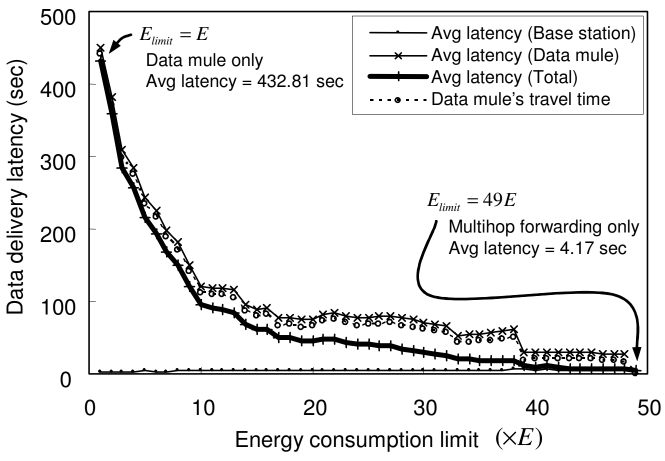
[TMF+07].

For ns2 simulator, we use FreeSpace propagation model with 100m communication range. We use 802.11 MAC (with RTS/CTS) with 2 Mbps raw bandwidth, which is the default value for ns2. Packet size is 400 Bytes.

Other parameters are set as follows. Energy consumption for sending/receiving unit data is assumed to be equal, i.e., $E_r = E_s$. The rate of data generation at each node λ_i is 100 Byte/sec. Let E denote the energy consumption at each node for "pure data mule" case without any node-to-node forwarding. Then E is expressed as $\lambda_i E_s$, and this is the minimum possible value of E_{limit}. We measured the latency for $E_{limit} = E, ..., 50E$. Effective bandwidth R is set to 400 Kbps, considering the overhead of RTS/CTS and packet header.

7.5.2 Results

Figure 7.3 is an example of forwarding plan and calculated path. The small circles represent the nodes and large circles are their communication ranges. Color of each circle represents how much data the node sends directly to the data mule. White circles mean zero and colored circles mean nonzero. As the forwarding algorithms try to gather data close to the base station, which is located in the center in this example, the nodes at the edge of the network forward all data to the ones closer to the center.



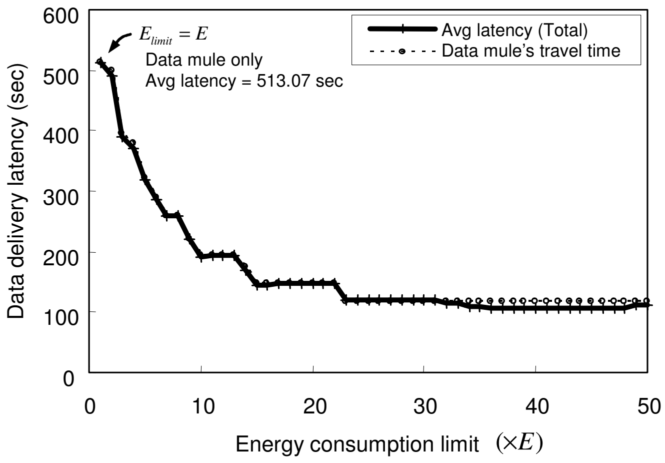
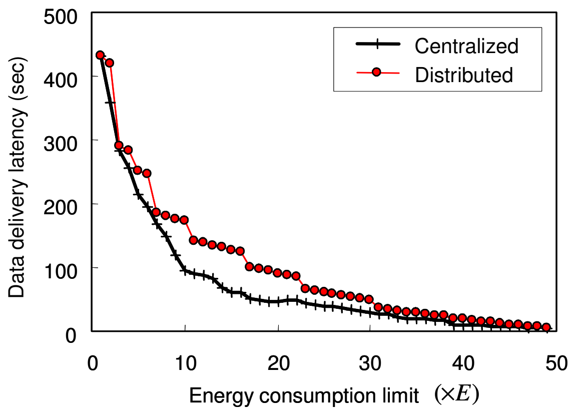


Figure 7.4: Data delivery latency for varying energy consumption limit: (top) connected network, (bottom) disconnected network. The centralized forwarding algorithm is used.

Nodes that have the base station within their communication range forward all the data directly to the base station.

In ns2 simulation, the delivery latency reached stability in all tested cases. Average number of periods until reaching stability was 5.2 (centralized) and 5.6 (distributed) in the connected network, and 4.0 (centralized) and 4.1 (distributed) in the disconnected network.

Figure 7.4 shows the simulation results for the connected network and disconnected network when the centralized algorithm is used for the forwarding problem. For both networks, $E_{limit} = E$ corresponds to the “pure data mule” case. The average latency in this case was 432.81 secs for the connected network and 513.07 secs for the



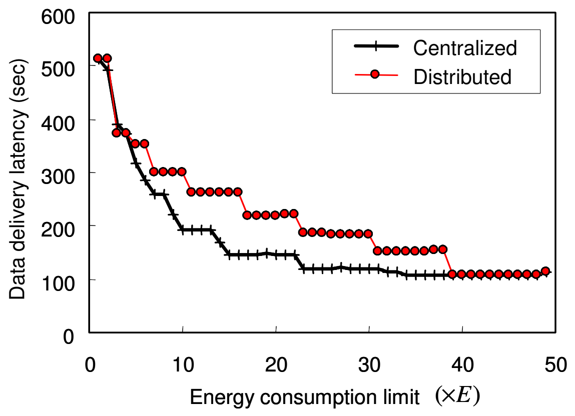


Figure 7.5: Average data delivery latency for different forwarding algorithms: (top) connected network, (bottom) disconnected network.

disconnected network. These are quite similar to the total travel time (440.73 secs and 511.68 secs, respectively). For the connected network, it became "pure multihop forwarding" when $E_{limit} = 49E$, where all the data are sent to the base station by multihop forwarding and the data mule is not used. The average latency in this case was 4.17 secs.

Figure 7.5 shows the comparison of average data delivery latency between the two forwarding algorithms. In both of the connected and disconnected networks, the centralized scheme based on LP formulation achieved shorter average latency than the distributed algorithm in most of the cases. On average, the ratio of average latency was 1.53 (min:0.93, max:2.05) for the connected network and 1.31 (min:0.95, max:1.80) for the disconnected network.

7.5.3 Discussions

For both the connected and disconnected networks, the simulation results showed the decrease of data delivery latency as the energy consumption limit increases. The decrease was almost monotonic, demonstrating fine-grained control of the trade-off between energy and latency. It was also shown that formulation of the forwarding problem, especially the choice of objective function is appropriate, due to the fact that the centralized algorithm achieved a better trade-off than the distributed one, which yields suboptimal forwarding plans.

We can also observe that the travel time of the data mule is nearly equal to the average latency for the data delivered by the data mule. It demonstrates that minimizing the travel time for the purpose of minimizing the data delivery latency is a valid approach. In addition, this implies that we can estimate the average delay solely by solving the DMS problem.

Figure 7.6 shows the histograms of data delivery latency for two different energy consumption limits for each of the connected and disconnected networks. As these figures show, regardless of the different network topology and the different total travel time, more than 98% of the data has delivery latency within double of the travel time. This means we can estimate the maximum delivery latency as well as the average.

In practice, our problem formulation and algorithms provide sensor network designers a good estimate of the data delivery latency when there is an energy consumption limit, which is imposed by their application scenarios. Conversely, since the energy-latency curve is nearly monotonic and the problem is solved in relatively short time[1], by using binary search, we can also estimate the maximum energy consumption at each node when there is a constraint on the average or the maximum of data delivery latency, as assumed in [XWXJ07].

7.6 Summary

In this chapter we presented the hybrid approach of data mule and multihop forwarding. The primary objective of combining these two approaches is to realize a flexible trade-off between energy consumption and data delivery latency. We have formulated the multihop forwarding as the Forwarding subproblem. Then we have designed a cen-

[1]For 100 nodes case, solving the forwarding problem and the DMS problem altogether takes around 10 secs on MATLAB running on a PC.

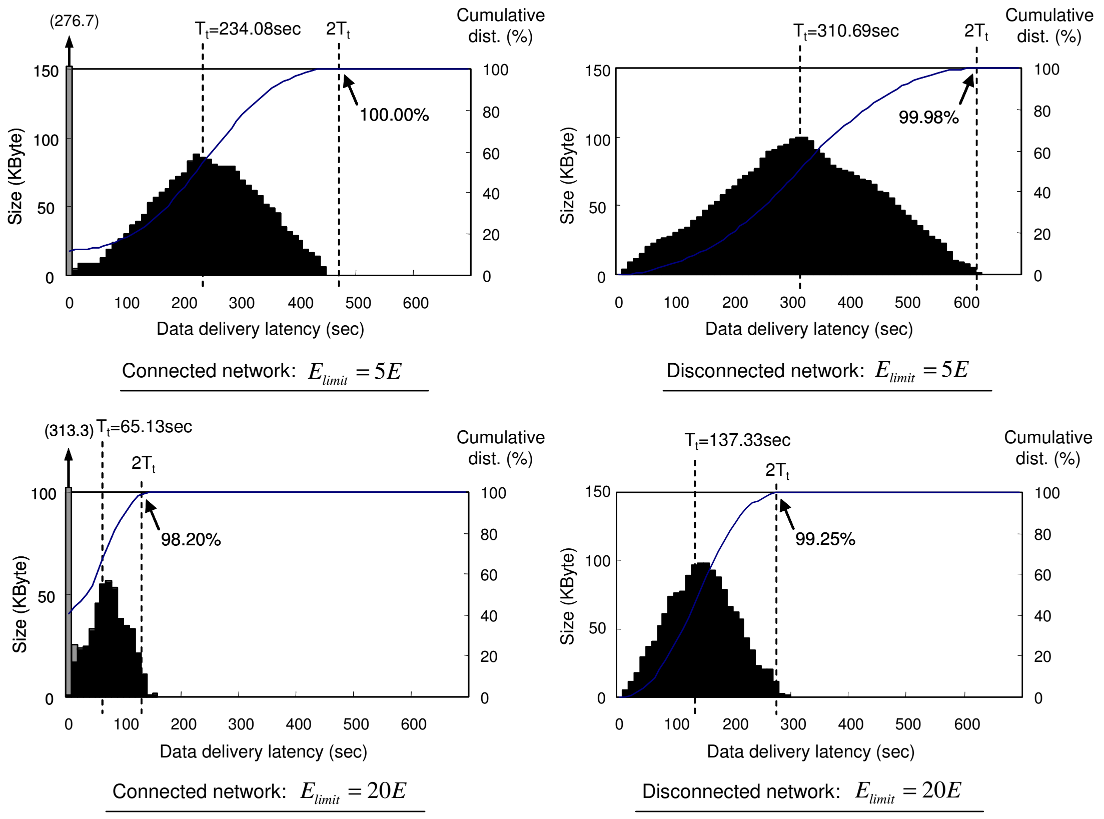


Figure 7.6: Histogram of data delivery latency: For the connected network, gray bars are for the data delivered to the base station from neighboring nodes, and black bars are for the data delivered via the data mule. The centralized forwarding algorithm is used.

tralized algorithm based on linear programming and also a distributed algorithm. We have also implemented the hybrid communication model and evaluated the performance by ns2. The results showed nearly monotonic decrease of the data delivery latency for larger energy consumption limit, demonstrating the effectiveness of the formulation and the algorithms in optimizing the energy-latency trade-off.

Acknowledgements This chapter, in part, has been published as “Optimizing Energy-Latency Trade-off in Sensor Networks with Controlled Mobility” by Ryo Sugihara and Rajesh K. Gupta in the proceedings of the 28th Annual Conference of the IEEE Communications Society (INFOCOM) Mini-conference [SG09c]. The dissertation author was the primary investigator and author of this paper.

Chapter 8

Extended DMS: Multiple Data Mules Case

In Chapter 8 and 9, we discuss two extensions of the DMS problem framework. In this chapter, we first discuss the case of multiple data mules. Using multiple data mules is a feasible option in some application scenarios to achieve better performance. We show how the DMS problem framework can be extended to formulate the problem of scheduling the motion and communication of multiple data mules. For 1-D case, we can formulate the problem as a scheduling problem with multiple processors. For 2-D case, path selection problem for multiple data mules is defined as the k-Label-Covering Tour problem by extending the Label-Covering Tour problem for single data mule case discussed in Chapter 6.

8.1 1-D DMS for Multiple Data Mules

In this section we discuss one dimensional DMS (1-D DMS) problem for multiple data mules case.

8.1.1 Problem Definition

First we define k-DM 1-D DMS by extending the 1-D DMS problem:

k-DM 1-D DMS
INSTANCE: Set $\mathcal{J}$ of location jobs, for each location job $\tau \in \mathcal{J}$, an execution time $e(\tau)$, and a set $\mathcal{I}^{(k)}(\tau)$ of feasible location intervals, for each feasible location interval $I^{(k)} \in \mathcal{I}^{(k)}(\tau)$, a release location $r(I^{(k)})$ and deadline location

$d(I^{(k)})$, a start $X_s^{(k)}$, a destination $X_d^{(k)}$, number of data mules K, and a constant T.
QUESTION: Is there a feasible speed control plan and a feasible job schedule for K data mules satisfying $T_k \leq T$ for all k, where T_k is the travel time of k-th data mule from $X_s^{(k)}$ to $X_d^{(k)}$?

Depending on the mobility assumption, there may be additional constraints (e.g., maximum speed, maximum acceleration) that determine the feasibility of a speed control plan. In the rest of the paper, we focus on an optimization version of the problem, in which we minimize $\max_k T_k$.

For the optimization problem, we can use other objective functions as well. One example is average travel time over all the data mules $\sum_k T_k/n$. However, it does not appropriately reflect the overall data delivery delay, for example when some of the data mules do not travel at all and have zero travel time. To avoid this, average travel time weighted by the amount of collected data ($\sum_k w_k T_k$) would be a more appropriate metric. However, this allows a small amount of data having a very large delay, which is also not preferable in some applications. It also has a practical issue that it is hard to be efficiently solved due to the non-linear cost function. Maximum travel time $\max_k T_k$ gives a guarantee on the data delivery latency, and appropriate for applications in which the maximum delivery latency is important. Although this is also a nonlinear function, it can be converted to a linear function using additional constraints, as we see later.

8.1.2 Basic Cases

First we discuss the basic cases, i.e., Constant speed and Variable speed cases. We consider two separate cases depending on whether the paths of data mules are identical or not.

For Identical Paths

When paths are identical for all data mules, feasible location intervals of each job are identical as well. We define the term "symmetric schedule" as follows. We call a schedule is "symmetric" when the speed control plan and job schedule are identical in all data mules. Then we have the following theorem:

Theorem 8.1.1. *For the optimization version of k-DM 1-D DMS with identical paths for Constant speed or Variable speed models, there exists an optimal symmetric schedule.*

Proof. We show there always exists an equal or faster symmetric schedule than asymmetric one. Consider splitting the total travel interval into short location intervals by dividing at a location which is either a release or deadline location of a job. Then, without loss of generality, we can assume each data mule moves at a constant speed in each of these short location intervals. For each of these intervals, let $v_1, v_2, ..., v_n$ denote the data mules' speed in the location interval. Without loss of generality, we assume $v_1 \leq v_2 \leq ... \leq v_n$. Total amount of collected data c satisfies $c \leq \sum_i l/v_i$, where l is the length of the location interval. Maximum time is l/v_1. Now, consider a symmetric schedule in which each data mule moves at $v' = n/(\sum_i 1/v_i)$, which is the harmonic mean of the original speed. Since $n(l/v') = \sum_i l/v_i \geq c$, it is possible to collect the same amount of data as in the original schedule. Maximum time is $l/v' \leq l/v_1$, where the equality is satisfied if and only if $v_1 = ... = v_n$. □

To find an optimal schedule, we first divide the execution time of each location job equally to each data mule and then apply the optimal algorithms for single data mule case.

For Arbitrary Paths

When the path of each data mule is different, we formulate the problem as a linear program in the following way. The formulation is based on the same idea as the single data mule case (Section 4.4.2), but we introduce additional variable z and constraints to convert the min-max objective into linear objective. For k-th data mule, we split the location interval $[X_s^{(k)}, X_d^{(k)}]$ into $(2m^{(k)} + 1)$ location intervals $[l_0^{(k)}(= X_s^{(k)}), l_1^{(k)}]$, $[l_1^{(k)}, l_2^{(k)}]$, ..., $[l_{2m}^{(k)}, l_{2m+1}^{(k)}(= X_d^{(k)})]$ $(l_i^{(k)} \leq l_{i+1}^{(k)})$, where $m^{(k)}$ is the number of feasible location intervals of all location jobs that are executable at this data mule, and each $l_i^{(k)}$ is either a release location or a deadline location. Let $z_i^{(k)}$ denote the time that the k-th data mule spends in location interval $[l_i^{(k)}, l_{i+1}^{(k)}]$, and $p_i^{(k)}(\tau)$ denote the time it allocates to job τ in this interval. Using a variable α to represent min-max objective, we have the following linear program:

Minimize α

Subject to

- (Min-max objective) $\forall k$, $\sum_{i=0}^{2m^{(k)}} z_i^{(k)} \leq \alpha$. Note that minimizing α is equivalent to minimizing the maximum of the left hand side over all k.

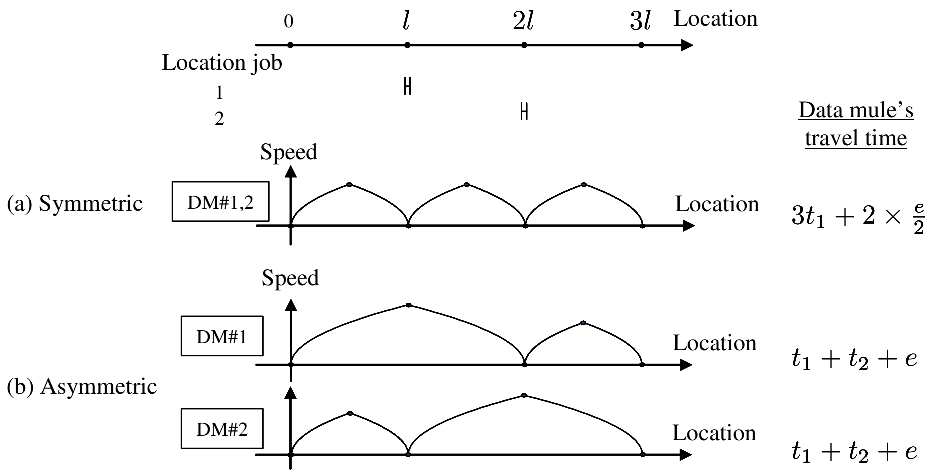


Figure 8.1: Example of non-optimal symmetric schedule (with acceleration constraint): Two location jobs have zero-length feasible location intervals with execution time e.

- (Max/min speed) $\forall k$, $0 \leq \forall i \leq 2m^{(k)}$, $v_{min} z_i^{(k)} \leq l_{i+1}^{(k)} - l_i^{(k)} \leq v_{max} z_i^{(k)}$ for variable speed model. For constant speed model, $(l_{i+1}^{(k)} - l_i^{(k)})/z_i^{(k)} = (l_{j+1}^{(k)} - l_j^{(k)})/z_i^{(j)}$ for all i, j satisfying $l_{i+1}^{(k)} - l_i^{(k)} > 0$, $l_{j+1}^{(k)} - l_j^{(k)} > 0$.
- (Positive allocation time) $\forall k$, $0 \leq \forall i \leq 2m^{(k)}$, $\forall \tau \in \mathcal{J}$, $p_i^{(k)}(\tau) \geq 0$.
- (Feasible intervals) $\forall k$, $0 \leq \forall i \leq 2m^{(k)}$, $\forall \tau \in \mathcal{J}$, if $[l_i^{(k)}, l_{i+1}^{(k)}] \notin \mathcal{I}^{(k)}(\tau)$, $p_i^{(k)}(\tau) = 0$.
- (Job completion) $\forall \tau \in \mathcal{J}$, $\sum_k \sum_{i=0}^{2m^{(k)}} p_i^{(k)}(\tau) = e(\tau)$.
- (Processor demand) $\forall k$, $0 \leq \forall i \leq 2m^{(k)}$, $\sum_{\tau \in \mathcal{J}} p_i^{(k)}(\tau) \leq z_i^{(k)}$.

8.1.3 General Case

When there is a constraint on acceleration (i.e., Generalized model), the problem for multiple data mules case is hard, as implied by the hardness of single data mule case.

One interesting observation is that, for identical paths case, symmetric schedule is not always optimal. Figure 8.1 shows such example. In this example, the data mules travel from location 0 to $3l$. Each of two location jobs (representing communication with two sensor nodes) have a zero-length feasible location interval at location l and $2l$, respectively. All curves in location-speed graphs represent acceleration/deceleration at the maximum rate a. Since $t_1 = 2\sqrt{l/a}$ and $t_2 = 2\sqrt{2}\sqrt{l/a}$, we have $3t_1 + e > t_1 + t_2 + e$ and thus the symmetric schedule is not optimal.

- Sort the jobs in the ascending order of the number of executable data mules.
  - For the jobs with the same number of executable data mules, sort them by execution time in the descending order (i.e., long job first).
- For all jobs, from head of the list,
  - Determine the strategy such that the maximum of current travel time is minimized. The strategy is one of the followings:
    - **Assign**: Assign the job to one data mule that can execute it.
    - **Spread**: Divide the job equally to all data mules that can execute it.
  - Remove τ from the list and update the travel time of each data mule.

Figure 8.2: Heuristic algorithm for k-DM 1-D DMS problem with acceleration constraint

We design a heuristic algorithm based on the idea similar to List Scheduling [Gra69]. Figure 8.2 shows the algorithm. First the jobs are sorted in decreasing order of the number of executable data mules. This is to assign the jobs that are executable at only one data mule first. In this way, there will be more freedom later in balancing the travel time of each data mule by appropriately allocating the jobs, which are executable at many data mules. The main idea of the algorithm is to assign jobs one by one to a data mule so that the maximum travel time is minimized. In addition to assigning a job to one of the data mules, we can also choose to spread the job to all data mules that can execute it by equally dividing the job's execution time, if the resulting maximum travel time is shorter than assigning the job to one data mule.

When each location job is not allowed to be executed by multiple data mules, we modify the heuristic algorithm by eliminating the "spread" option when determining the strategy. This modified version of the algorithm is also applicable to the case without acceleration constraint. Although it is not optimal anymore when applied to constant speed or variable speed cases, it is appropriate when it is infeasible for each node to communicate with multiple data mules.

8.2 Path Selection for Multiple Data Mules

In this section, we first define the path selection problem for multiple data mules case and then present an approximation algorithm. We also present an integer linear program (ILP) formulation of the problem and apply relaxations in several ways to obtain the lower bounds. In the end we present the results from simulation experiments.

8.2.1 Problem Definition

Based on the Label-Covering Tour problem[1] for single data mule case, we define k-Label-Covering Tour (k-LCT) problem for k data mules case as follows. We are given an undirected complete graph $G = (V, E)$ where each vertex in $V = \{v_0, v_1, ..., v_n\}$ is a point in $\mathbf{R}^2$, a cost function on edges $c : E \to \mathbf{Q_0^+}$, a set $L = \{l_1, ..., l_n\}$ of labels, and constants $r \in \mathbf{Q_0^+}$ and $K \in \mathbf{Z}^+$. Each edge $e_{ij} \in E$ is associated with subset $L_{ij} \subseteq L$. For $p = 1, ..., n$, $l_p \in L_{ij}$ iff the Euclidean distance between v_p and edge e_{ij} is equal to or less than r. A subtour T is a list of subset of all vertices that starts and ends with v_0, allowing multiple visits to each vertex. A set of subtours $\{T_1, T_2, ..., T_K\}$ is "label-covering" when it satisfies at least one of the followings for $p = 1, ..., n$: 1) $\exists k, e_{ij} \in T_k(E), l_p \in L_{ij}$, where $T_k(E)$ is the set of edges traversed by T_k, or 2) $dist(v_0, v_p) \leq r$, where $dist(v_i, v_j)$ is the Euclidean distance between v_i and v_j. Find a set of label-covering subtours $\{T_1, T_2, ..., T_K\}$ that minimizes the maximum of cost of subtours $\max_k \sum_{e_{ij} \in T_k(E)} c(e_{ij})$.

As in the previous section, we focus on the case that c is the Euclidean distance.

8.2.2 Approximation Algorithm

Our strategy in designing an approximation algorithm is to solve TSP for k salesmen (k-TSP) first and then to shortcut each subtour so that the label-covering property is maintained. For solving k-TSP problem, we use k-SPLITOUR algorithm [FHK78]. k-SPLITOUR algorithm constructs k subtours by splitting 1-TSP tour in the following way:

- Find a 1-TSP tour $R = (v_0, v_1, ..., v_n, v_0)$ with $\sum_{e \in R(E)} c(e) = D$.
- For each $j, 1 \leq j < k$, find the last vertex $v_{p(j)}$ such that the cost of the path from v_0 to $v_{p(j)}$ along R is not greater than $(j/k)(D - 2c_{max}) + c_{max}$, where $c_{max} =$

[1]Hereafter we call it the "1-LCT" problem for clarity.

```
• Find $k$-TSP subtours $\{R_1, R_2, ..., R_k\}$ using $k$-SPLITOUR algorithm.
• While true,
    – For each subtour $R$, in the decreasing order of total cost,
        ∗ Find a visited vertex $v$ s.t.
            · the label-covering property is maintained if $v$ is skipped by $R$, and
            · $c(R) - c(R\backslash v)$ is maximized.
        ∗ If $v$ is found, $R \leftarrow R\backslash v$ and break the inner loop; Otherwise continue.
    – If no vertex was skipped in all subtours, stop and output the subtours.
```

Figure 8.3: Approximation algorithm for k-LCT problem

$\max_i c(e_{0i})$.

- Form k subtours as $R_1 = (v_0, v_1, ..., v_{p(1)}, v_0), R_2 = (v_0, v_{p(1)+1}, ..., v_{p(2)}, v_0)$, $..., R_k = (v_0, v_{p(k-1)+1}, ..., v_n, v_0)$.

We have the following theorem about the approximation ratio:

Theorem 8.2.1 (Frederickson et al. [FHK78]). *If $\hat{C}_k$ is the cost of the largest of the k subtours generated by k-SPLITOUR algorithm, and C_k^* is the cost of the largest subtour in an optimal solution of k-TSP, then*

$$\hat{C}_k / C_k^* \quad \leq \quad \alpha + 1 - 1/k,$$

where α is the bound for the single traveling salesman algorithm.

Figure 8.3 shows the algorithm for the k-LCT problem. In the algorithm, we first use k-SPLITOUR algorithm to construct k subtours. Then we apply shortcutting for each subtour as long as the label-covering property is not violated. Shortcutting is attempted on the longest subtour first. If not successful, we try the second longest one, and the third one, etc., until there is no subtours that can be shortcutted. We have the following guarantee on the approximation ratio:

Theorem 8.2.2. *If APP is the approximate solution of a given instance of the k-LCT problem and OPT is the optimal one, $APP \leq (\alpha + 1 - \frac{1}{k})(OPT + 2nr)$, where α is the approximation ratio of TSP algorithm and r is the communication range.*

Proof. As we construct k-label-covering subtours by shortcutting k-TSP subtours, we have $APP \leq \hat{C}_k$. By definition of the label-covering property, for any unvisited vertex there exists an edge in k-label-covering subtour within distance c. Thus, by using the same technique for Theorem 6.3.1 to convert a label-covering tour into a TSP tour, we have $C_k^* \leq OPT + 2nr$. Then, from Theorem 8.2.1, the theorem follows. □

8.2.3 Integer Linear Program Formulations

First we give an ILP formulation of the 1-LCT problem. Variables are

- $x_{ij} \in \{0, 1\}$: edge $e_{ij} = (v_i, v_j)$ is included in the tour iff $x_{ij} = 1$
- $y_i \in \{0, 1\}$: node v_i is visited iff $y_i = 1$

Constants are

- $c_{ij} \in \mathbf{Q}_0^+$: cost of edge e_{ij}
- $d_{i,pq} \in \{0, 1\}$: equals 1 iff node v_i is within the distance r from edge e_{pq}.

Then the 1-LCT problem is

Minimize $\sum_{i,j} c_{ij} x_{ij}$

Subject to

$$x_{ii} = 0 \ (\forall i) \quad , \quad y_0 = 1 \tag{8.1}$$

$$y_i \leq \sum_j x_{ji} \ = \ \sum_j x_{ij} \leq (n-1) y_i \quad \forall i \tag{8.2}$$

$$\sum_{p,q} x_{pq} d_{i,pq} \ \geq \ 1 + y_i \quad \forall i \geq 1 \tag{8.3}$$

$$\frac{1}{|S|} \sum_{i \in S} y_i \leq \sum_{i \in S, j \notin S} x_{ij} \ \leq \ \sum_{i \in S} y_i \quad \forall S \subseteq V \backslash \{v_0\} \tag{8.4}$$

$$\frac{1}{|S|} \sum_{i \in S} y_i \leq \sum_{i \in S, j \notin S} x_{ji} \ \leq \ \sum_{i \in S} y_i \quad \forall S \subseteq V \backslash \{v_0\} \tag{8.5}$$

Inequality (8.2) enforces in- and out-degree of each vertex to be equal. It also enforces that both degrees are zero when the vertex is not visited. We obtain this by combining the following two constraints:

- if $y_i = 0$, $\sum_j x_{ji} = \sum_j x_{ij} = 0$
- if $y_i = 1$, $\sum_j x_{ji} = \sum_j x_{ij}$

Inequality (8.3) is the label-covering property. It is obtained by combining the following two constraints:

- if $y_i = 0$, $\sum_{p,q} x_{pq} d_{i,pq} \geq 1$
- if $y_i = 1$, no constraint ($\sum_{p,q} x_{pq} d_{i,pq} \geq 2$ is trivially satisfied)

Finally, inequalities (8.4) and (8.5) are the constraints for eliminating invalid subtours. They are obtained by combining the following two constraints:

- if $\sum_{i \in S} y_i = 0$, $\sum_{i \in S, j \notin S} x_{ij} = \sum_{i \in S, j \notin S} x_{ji} = 0$
- if $\sum_{i \in S} y_i > 0$, $\sum_{i \in S, j \notin S} x_{ij} \geq 1$ and $\sum_{i \in S, j \notin S} x_{ji} \geq 1$

Note that these subtour constraints consist of exponential number of inequalities. However, we can use the cutting-plane technique (e.g., [Pat03]), in which we solve the ILP problem without these constraints first, add only violated inequalities and solve again, and repeat this until we obtain the tour without invalid subtours.

We can easily extend this formulation to the k-LCT problem in the following way. Instead of x_{ij}, y_i, the variables are $x_{ij}^{(k)}$ and $y_i^{(k)}$, representing whether edge e_{ij} is included in k-th tour (i.e., tour of k-th data mule) and whether vertex v_i is visited by k-th tour, respectively. To allow the min-max objective, we have an additional variable z. Constants are the same as in the 1-LCT problem. Then, k-LCT problem is to minimize z subject to $\forall k. \sum_{i,j} c_{ij} x_{ij}^{(k)} \leq z$ and inequalities (8.1)-(8.5), with substituting $x_{ij}^{(k)}$, $y_i^{(k)}$ for x_{ij}, y_i.

8.2.4 Obtaining Lower Bounds

The ILP problem above cannot be solved in a realistic time when the number of variables is large. Instead, we use that for obtaining lower bounds of the optimal solution by applying various relaxations.

Lower bounds are useful because of the following reason. In Theorem 8.2.2, we have obtained a theoretical guarantee on the performance of the approximation algorithm for the k-LCT problem. However, the upper bound given in the theorem is loose when r is large. This gives a motivation for evaluation by experiments, in which we compare the approximate solution with lower bounds to figure out the performance in practical settings.

We consider the following relaxations to obtain the lower bounds:

- **ILPcover**: ILP without subtour constraints: This finds subtours that collectively satisfy the label-covering property. Note that the number of subtours is arbitrary and some of them may not include v_0.
- **LPCP**: LP relaxation + cutting plane: first solve LP relaxation of the original ILP with the subtour constraints for $|S| = 2$ case. Then, cutting plane method is applied for at most ten times.
- **MaxCost**: Trivial lower bound by $2\max\{\max_i\{c_{0i}\} - r, 0\}$. This is the smallest possible cost to touch the communication range of the farthest node.

We need a different approach to use cutting plane method in **LPCP**. In the original ILP formulation, we just needed to find invalid subtours in an intermediate solution, add subtour constraints for them, and repeat that until we find a valid solution. However, this does not work in **LPCP** because intermediate solutions are generally fractional in the LP relaxation and thus cannot identify invalid subtours directly. Instead, we regard the value of $x_{ij}^{(k)}$ as the weight of edge (i, j) and add the subtour constraints for the cycles with large mean weight.

Finding a cycle that has the maximum mean weight in a graph is done in polynomial time [Kar78]. We modify the algorithm to find only the cycles with length at least three and applied it iteratively by eliminating the vertices in the cycle. Subtour constraints are added if they were not added previously. If no new invalid subtours were found, or it reached the maximum number of iterations (set to 10), the solution at that point is used.

The following theorem enables us to obtain a lower bound for k-DM case by "scaling" the result for 1-DM case:

Theorem 8.2.3. *For a given graph G and a lower bound LB_1 of the 1-LCT problem for G, $LB_1/k \leq OPT_k$, where OPT_k is the optimal solution for the k-LCT problem for G.*

Proof. From any set of k-LCT subtours, by connecting each subtour, we can make a 1-LCT tour. If there exists a set of k-LCT subtours whose maximum length is strictly less than OPT_1/k, the 1-LCT tour made by connecting these subtours have length strictly less than OPT_1, which is a contradiction. Therefore we have $LB_1/k \leq OPT_1/k \leq OPT_k$. □

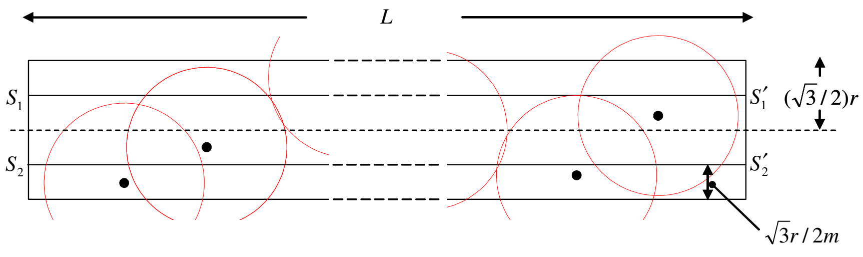


Figure 8.4: Randomly generating test cases: An example with two data mules is shown.

8.3 Performance Evaluation

We evaluate the performance of our algorithms in realistic situations by simulation experiments. We first compare the path length by the proposed algorithm with two other strategies and also with the lower bounds. Then, we compare the travel time in these strategies by solving the 1-D problem.

8.3.1 1-D DMS

Method

We use MATLAB for simulation experiments. Besides the heuristic algorithms, we have implemented a naive method for speed control. In the naive method, a data mule stops to collect data from each node one by one. Specifically, a data mule moves to the point on the path that is the closest to the node, collects data from the node while stopping, and moves to the next point. This is similar to the one used in [MY07], though they do not assume an acceleration constraint. The difference from the experiments in Section 5.6.1 is that there are multiple data mules this time. Each node is assigned to the data mule whose path is the closest to it.

We have random nodes only and no fixed nodes (Figure 8.4). The vertical coordinate of the deployment area is $[-(\sqrt{3}/2)r, (\sqrt{3}/2)r]$, which is same as the first case. The path for i-th data mule is a horizontal line on the vertical coordinate of $((2i-1)\sqrt{3}/2k - \sqrt{3}/2)r$, where k is the total number of data mules. This setting makes each data mule cover the strip of same size. We used the same parameters as the first experiment: $n = 50$, $r = 100[m]$, and $L = 2100[m]$. The number of data mules is $k = 1, 2, 3$.

We set the execution time $e = 10[sec]$. Each experiment is repeated for 10 times on each case and the average is used as the result. We use the variable speed with acceleration constraint as the mobility model of data mules and set $a_{max} = 1[m/s^2]$ and $v_{max} = 10[m/s]$ to roughly simulate the mobility capability of a radio controlled helicopter used in [TMF+07].

Results

Next we compare the heuristic algorithm and the naive method in multiple data mules setting. Since the energy consumption at the nodes measured by the amount of transmission is identical in both methods, we compare the total travel time.

Figure 8.5 shows examples of speed control plans from these two methods. For random node placement as shown in Figure 8.5(a), we use two data mules that move along two dotted horizontal lines. Figure 8.5(b) shows the speed changes of two data mules in the heuristic algorithm. The travel time of the data mules are 264.55 sec and 266.22 sec, respectively. Figure 8.5(c) shows the ones in the naive method, in which the travel time is 727.75 sec and 630.28 sec, respectively. Not only the maximum travel time is much shorter in the heuristic algorithm, travel time is also balanced between two data mules compared to the naive method.

Figure 8.6 shows the travel time for different number of data mules. For each of $k = 1, 2, 3$ case, the average of maximum travel time is 510.99 sec, 275.50 sec, 228.93 sec for the heuristic algorithm and 1082.69 sec, 725.54 sec, 609.51 sec for the naive method, respectively. The heuristic algorithm reduced the travel time by 50-60% from the naive method on average.

8.3.2 Path Selection

We evaluate the performance of the approximation algorithm in Figure 8.3 in realistic situations by simulation experiments. We first compare the path length by the proposed algorithm with two other strategies and also with the lower bounds. Then, we compare the travel time in these strategies by solving the 1-D DMS problem.

Method

We use MATLAB for simulation. For simulations, nodes are deployed at random locations in a circular area with the base station at the center. Number of nodes is 40

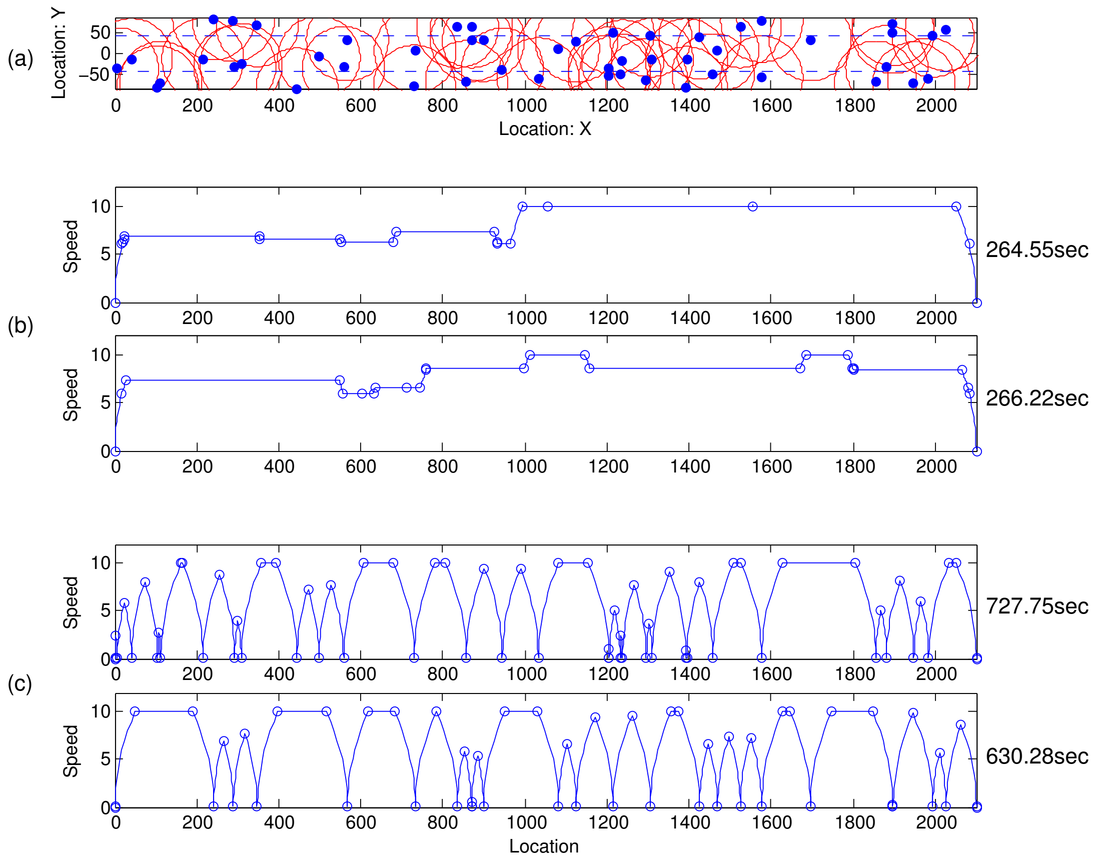


Figure 8.5: Example of speed control plans (Number of data mules $k = 2$): (a) Node placement. Filled circles are the node locations and large circles represent communication ranges. Two dotted lines correspond to the trajectories of two data mules; (b) Speed control plans from the heuristic algorithms; (c) Speed control plans from the naive method.

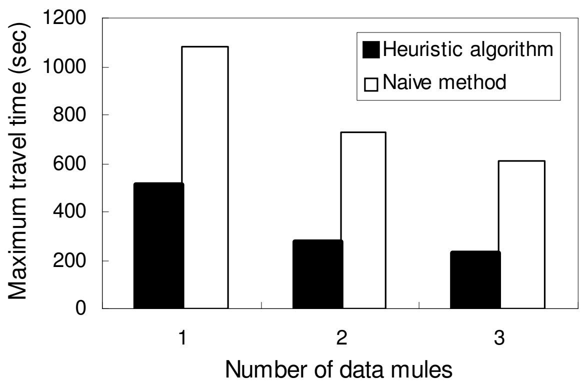


Figure 8.6: Travel time for multiple data mules case: Maximum of all data mules. Average of 10 experiments.

and radius of deployment area is fixed to $d = 600[m]$. We compute our results as the average of ten different node deployments. The number of data mules is $k = \{1, 2, 3, 4\}$ and communication range is $r = \{0, 50, 100, 150\}[m]$. Euclidean distance is used as the cost function c. We use Concorde TSP solver [Con] to find an optimal TSP tour. For the 1-D DMS problem, we set maximum acceleration of data mule a_{max} to $1[m/s^2]$ and maximum speed v_{max} to $10[m/s]$. Execution time is $e = \{10, 30, 60\}[sec]$ for each node. We use the heuristic algorithm (presented in Section 5.4) to solve the 1-D DMS problem under acceleration constraint.

We have implemented the following three strategies for comparison:

- **Proposed**: Use k-LCT subtours obtained by the approximation algorithm (Figure 8.3).

- **Overlay**: Use a 1-TSP tour for all k data mules and each data mule collects equal amount of data from each node, i.e., symmetric schedule on identical paths. This models the SIRA (single-route algorithm) strategy in [ZAZ05].

- **Partition**: Use k-TSP subtours and each data mule collects data only from the nodes on the subtour it is assigned. This models the MURA (multi-route algorithm) strategy in [ZAZ05].

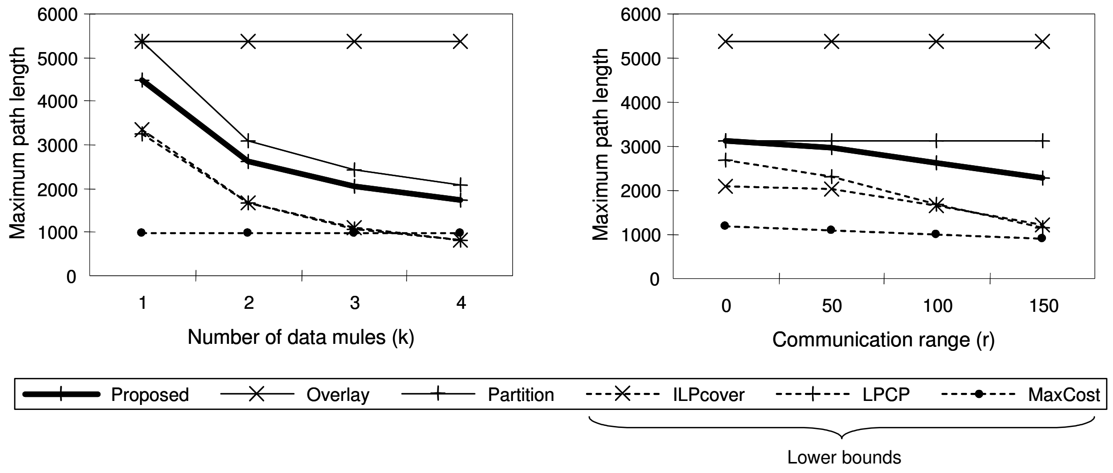


Figure 8.7: Maximum path length: (left) For different k. Communication range is fixed to $r = 100$; (right) For different r. Number of data mules is $k = 2$. Data for ILPcover ($k = 2, 3, 4$) and LPCP ($k = 3, 4$) are due to the scaling of $k = 1$ case.

Results

Figure 8.7 shows the maximum path length for different number of data mules k and different size of communication range r. When k is changed, the path length does not change for the overlay strategy, since it always uses 1-TSP tours regardless of k. Both of the partition strategy and the proposed strategy scale well with the number of data mules. When the size of communication range r is changed, the path length did not change in either the overlay or partition strategies, because both of them use (k-)TSP tours. In the proposed strategy, the length decreased for larger communication range.

In comparing with the lower bounds, the average ratios of approximate solutions to the lower bounds are less than two in the tested cases. We cannot guarantee anything from these results, since we do not know how tight the lower bounds are and how bad the ratio can be in other parameters. Nevertheless, this is useful information to give estimates to the practical performance, since the bound by Theorem 8.2.2 is very loose in many of the tested cases.

Figure 8.8 compares the maximum travel time of the three strategies. We have tested three different execution time e to see the effect of the communication bandwidth and/or the amount of data in each sensor node. In $e = 10$ case, the proposed strategy yielded up to $30 - 40\%$ shorter travel time than other two strategies. As e increases, which corresponds to less communication bandwidth or larger amount of data in sensor

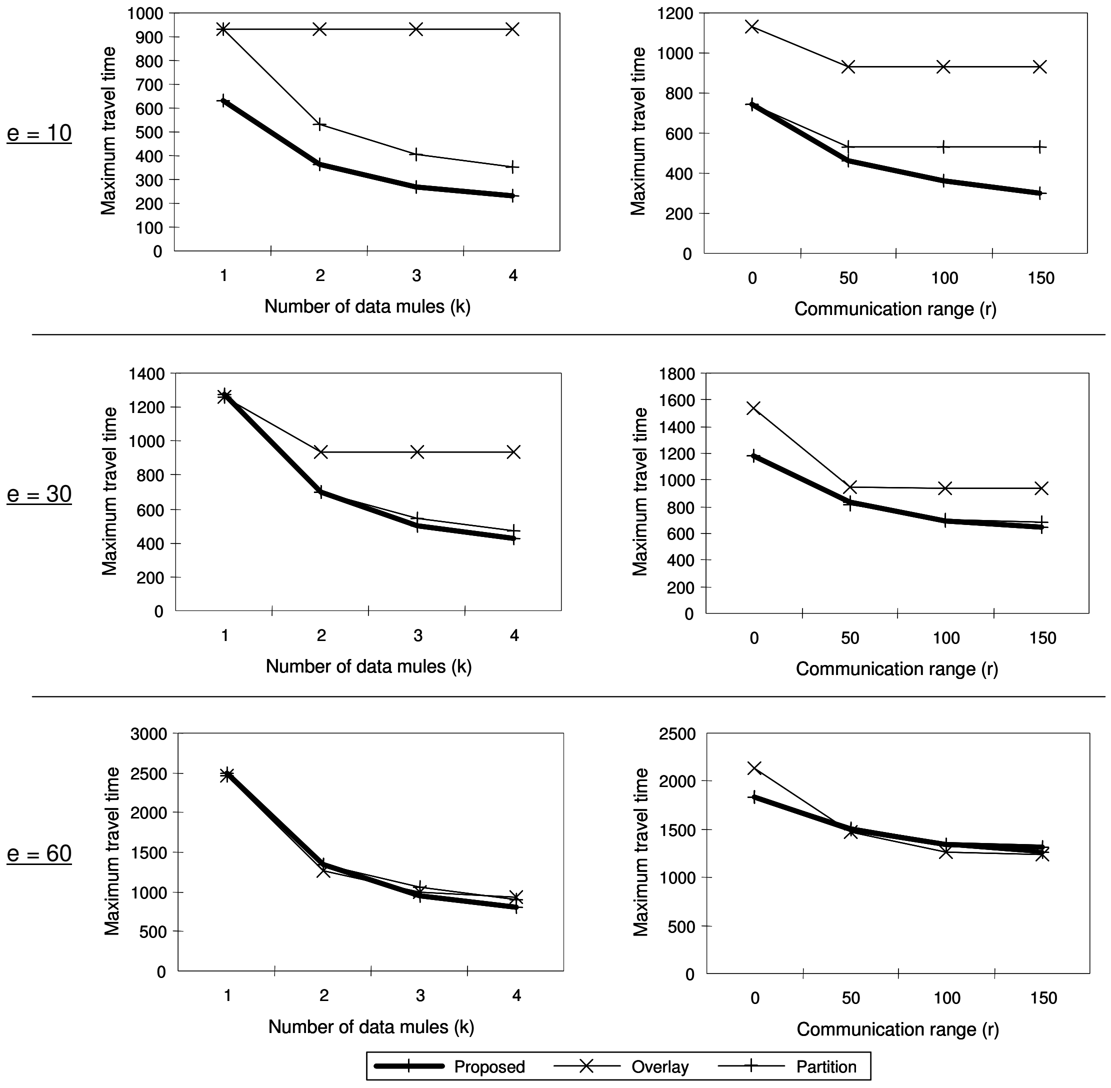


Figure 8.8: Maximum travel time when execution time $e = \{10, 30, 60\}$. (left) For different k. Communication range is fixed to $r = 100$; (right) For different r. Number of data mules is $k = 2$.

nodes, the differences between the strategies shrank. This is because the execution time becomes more dominant in determining the travel time than the path length.

8.4 Related Work

In case of multiple data mules, notable works include Jea et al. [JSS05], who present a distributed coordination scheme for allocating sensor nodes to each data mule to achieve a good distribution of communications load. They assumed fixed paths for each data mule and no speed control. Our work is complementary to their work and focused on identifying the performance limits of centralized schemes when we can freely determine the path and speed of each data mule. This would serve as a performance target for distributed schemes. Zhao et al. [ZAZ05] present heuristic algorithms for choosing the paths of multiple "message ferries" for communication among stationary nodes. Somasundara et al. [SRS07] present a heuristic algorithm for path selection based on the formulation as a vehicle routing problem (VRP). These work do not address speed control problem, and we extend their work by considering both path and speed of data mules to optimize the data delivery latency in multiple data mules environment.

Based on the connections between the 1-D DMS problem and speed scaling problem, the multiple data mules case corresponds to multiprocessor speed scaling (e.g., [AMS07]). The 1-D DMS problem is simpler in some ways than the DVS problems due to the assumptions such as preemptible and malleable[2] jobs. On the other hand, acceleration constraint is not generally studied in the scheduling literature.

Path selection problem for multiple data mules case is similar to TSP with k salesmen (k-TSP) and vehicle routing problem (VRP). For k-TSP, we have discussed an approximation algorithm for the case with min-max objective [FHK78]. There are many variations of VRP and approximation algorithms for some of them are presented in [AHL06]. One of the variations most similar to our formulation of the k-Label-Covering Tour is the single-depot case allowing split delivery (e.g., [AS08]). Different from the normal VRP, each customer can be visited more than once in the split delivery VRP. However, one of the characteristic features of k-LCT problem is that we assume that each node has circular range around it where the communication is possible. On the other hand, formulations based on the VRP would be useful for the DMS problem if we

[2]In job scheduling terminology, "malleable jobs" can be allocated to arbitrary number of processors.

employ the assumptions such as that each data mule has a capacity.

8.5 Summary

In this chapter we have shown how we can extend the DMS problem framework to multiple data mules case. For the 1-D case, in which speed control plan and communication schedule of each data mule are to be determined when the paths are given, we have designed optimal and heuristic algorithms for the mobility models with or without acceleration constraint. For the 2-D case, i.e., path selection problem, we designed an approximation algorithm and also formulated the problem as an integer linear program for obtaining lower bounds through relaxations. Simulation experiments in practical settings demonstrated that the proposed algorithms improve data collection time by up to $30-40\%$ compared to earlier work on Message Ferrying.

Acknowledgements This chapter, in part, has been submitted for publication as "Path Planning of Data Mules in Sensor Networks" by Ryo Sugihara and Rajesh K. Gupta in ACM Transactions on Sensor Networks [SG09d]. The dissertation author was the primary investigator and author of this paper.

This chapter, in part, has been submitted for publication as "Speed Control and Scheduling of Data Mules in Sensor Networks" by Ryo Sugihara and Rajesh K. Gupta in ACM Transactions on Sensor Networks [SG09e]. The dissertation author was the primary investigator and author of this paper.

Chapter 9

Extended DMS: Partially-known Communication Ranges

In this chapter we discuss another extension of the DMS problem framework. Specifically we use a different wireless connectivity model that is based on more realistic radio environments. We first design a realistic connectivity model, which we call the "hybrid model." In this model, communication range is known only partially and consists of two parts: known and unknown communication ranges. Under this model, we formulate the problem as a semi-online scheduling problem, in which partial information about jobs is available offline and more information gets available at runtime. We discuss both non-periodic and periodic cases and design semi-online algorithms. Finally we implement these algorithms on MATLAB and ns2 and evaluate the performance by simulation experiments.

9.1 Hybrid Connectivity Model

A simple connectivity model is the circular, fixed-range model. In this model, received signal strength at distance is estimated by considering mean pass loss, and the radius of the circle is determined by thresholding the strength. Despite its appearance in many papers, it has been pointed out that this model does not reflect the reality of the wireless channel [GKW+02, KNE03].

In contrast, probabilistic models takes shadowing and fading into account. While these are more realistic, they make the problem more difficult since there is a finite

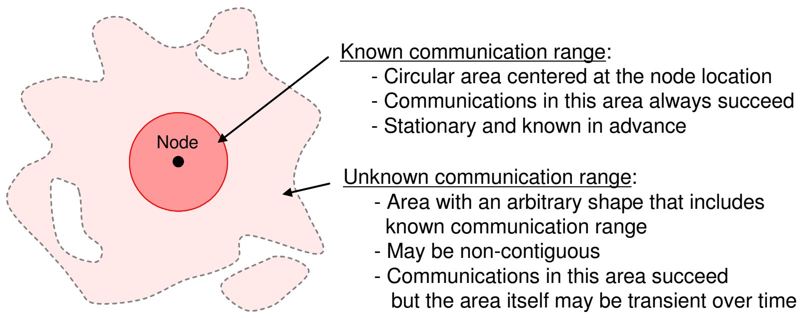

Node
Known communication range:
- Circular area centered at the node location
- Communications in this area always succeed
- Stationary and known in advance
Unknown communication range:
- Area with an arbitrary shape that includes known communication range
- May be non-contiguous
- Communications in this area succeed but the area itself may be transient over time

Figure 9.1: Hybrid connectivity model

probability of communication failure. In combinatorial frameworks used for solving the path and scheduling problems, this results in excessively conservative assumptions such as "communication is possible only at the exact location of each node" as in [SRS07, XWJL08] or "no knowledge about connectivity" as in [SKJ+06]. Furthermore, recent study shows empirically that probabilistic models are still insufficient to model many realistic aspects [LCL07]. Specifically, in the Shadowing model implemented in ns2 [ns2], noises are assumed spatially independent and follow Gaussian, both of which are not true in real environments.

Our hybrid connectivity model is based on the observation that, even though the connectivity at distance fluctuates dramatically, there is a certain range in the vicinity of node that we can guarantee the connectivity for sure. Figure 9.1 shows the idea of the hybrid connectivity model. We have a fixed circular range called *known communication range* around the node. A larger range that contains known communication range is *unknown communication range*. Note that the communication is always successful in both known and unknown communication ranges. The uncertainty at distance is represented as the temporal change of unknown communication range. We do not make any assumptions about the deterministic or probabilistic characteristics of the unknown communication range, except that it always contains the known communication range. The size of known communication range needs to be chosen accordingly, depending on the degree of uncertainty of channel and the environments (indoor/outdoor, terrain, etc.).

Having the known communication range enables the offline algorithm that has a performance guarantee. On the other hand, the performance is largely affected by the size of communication range, as shown in Section 6.2. Thus the opportunity that the offline algorithm misses becomes huge under a more uncertain environment, because there is larger portion of unknown communication range that the offline algorithm cannot exploit. This motivates us to consider semi-online algorithms, which are based on offline plans based on partial knowledge and try to improve at runtime with new information.

9.2 Semi-Online Scheduling: Non-Periodic Case

Under the hybrid connectivity model consisting of known and unknown communication ranges, we design two different semi-online algorithms. While we seek to exploit the unknown communication range to minimize the performance degradation in highly uncertain environments, our design goal is to guarantee that the proposed algorithm never produces a result worse than the offline algorithm. This is not trivial since online algorithms usually tend to produce results that are theoretically bounded in quality that is worse than the worst case offline algorithms.

9.2.1 Preliminaries

Here are some assumptions. All the sensor nodes except the data mule are stationary. The location of each sensor and the amount of data it has are known. There is a single base station where the data mule starts from and travels back after collecting data from the sensor nodes. Communication bandwidth is known and thus the time it takes to collect data from each node is calculated. The hybrid connectivity model is assumed. The radius r_K of known communication range is same for all nodes and is known. The unknown communication range is not known offline, but the data mule can tell whether its current location is in the unknown communication range of each node (i.e., whether the data mule can collect data from that node)[1]. There is only one data mule. The data mule knows its current location and moves along a polygonal path starting and ending at the base station. The data mule can change the speed in the range of $[0, v_{max}]$ without any acceleration constraint.

We use P_{Off} and S_{Off} to denote the path and schedule by the offline algorithm, respectively. A schedule is represented as a set of schedule entry $s = (I, job, v)$, where $s(I)$ is the location interval on the path, $s(job)$ is the job to execute (i.e., node to collect

[1]This is a strong assumption and will be removed when we design a communication protocol and do simulations in ns2.

data from), and $s(v)$ is the speed of data mule. The travel time by schedule S_{Off} is denoted as T_{Off}.

9.2.2 1-D Semi-Online Algorithm

The 1-D semi-online algorithm is a rather straightforward extension of the offline algorithm. It uses the same path and the schedule as the offline algorithm and tries to do opportunistic data collection while the data mule is idle.

Algorithm 4 shows the pseudocode for the 1-D semi-online algorithm. The data mule makes decision at each location along the path. When there is an offline schedule entry that includes the current location and the job is not finished yet, the data mule executes it (Line 3-5). Otherwise, the data mule moves at the maximum speed and does opportunistic data collection (Line 7-8). The schedule entries for finished jobs are eliminated accordingly (Line 10).

Algorithm 4 1-D semi-online algorithm

```
 1: $S \leftarrow S_{Off}$                                  ▷ Copy offline schedule
 2: while moving along $P_{Off}$ do
 3:     if $\exists s \in S.\ x \in s(I)$ then             ▷ $x$: current location
 4:         $v \leftarrow s(v)$
 5:         execute $s(job)$
 6:     else
 7:         $v \leftarrow v_{max}$
 8:         Execute any available jobs
 9:     end if
10:     Eliminate all $s \in S$ s.t. $s(job)$ is finished
11: end while
```

When the data mule collects data opportunistically and there are multiple nodes that it can communicate, there is a problem of how to choose a node. It can be arbitrary, but one possible policy that we use in the numerical experiments later is to give higher priority based on the order in the TSP tour constructed when determining the offline label-covering tour.

We have the following theorem that guarantees the travel time is shorter than that of the offline algorithm:

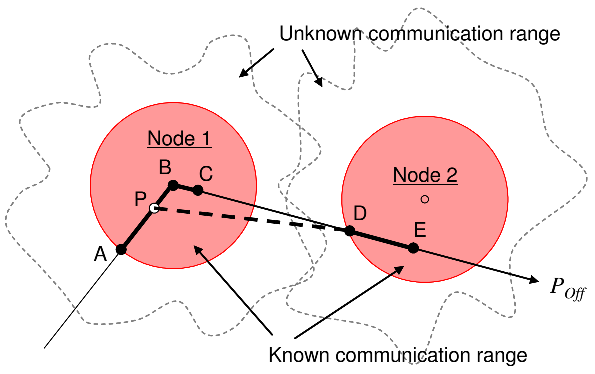


Figure 9.2: 2-D semi-online algorithm: If the data collection from Node 1 finishes at P, the data mule directly heads for D, where the next offline schedule entry starts.

Theorem 9.2.1. $T_{1D} \leq T_{Off}$

Proof. Let $\mathcal{I}_{idle}, \mathcal{I}_{busy}$ denote the sets of location intervals that the data mule is idle and busy in the offline schedule, respectively. For the offline schedule, let $T_{Off}^{(idle)}, T_{Off}^{(busy)}$ denote the total travel time for $\mathcal{I}_{idle}, \mathcal{I}_{busy}$. Similarly, define $T_{1D}^{(idle)}, T_{1D}^{(busy)}$ for the 1-D semi-online algorithm.

In the 1-D semi-online algorithm, the data mule may collect some data in $\mathcal{I}_{idle}$. Thus the amount of data collected in $\mathcal{I}_{busy}$ in the semi-online algorithm is at most as much as that in the offline algorithm. Since the data mule moves at the maximum speed after finishing data collection, $T_{1D}^{(busy)} \leq T_{Off}^{(busy)}$. Since the data mule moves at the maximum speed in $\mathcal{I}_{idle}$ both in the semi-online algorithm and in the offline algorithm, $T_{1D}^{(idle)} = T_{Off}^{(idle)}$. Therefore, $T_{1D}^{(busy)} + T_{1D}^{(idle)} \leq T_{Off}^{(busy)} + T_{Off}^{(idle)}$ and the theorem follows. □

9.2.3 2-D Semi-Online Algorithm

Algorithm 5 shows the pseudocode for the semi-online algorithm. At first the data mule follows the offline path and schedule, and does opportunistic data collection when there is no schedule entry to execute at the current location (Lines 3-13). When one of the jobs finishes, the data mule takes a shortcut to the location where the next schedule entry starts, moving at the maximum speed (Lines 16-20). Figure 9.2 explains this. In the figure, the bold lines (A-B-C, D-E) on the offline path P_{Off} represent the intervals covered by offline schedule entries. Now, in the semi-online algorithm, assume that the data collection from Node 1 finished at P, due to that some of the data have

Algorithm 5 2-D semi-online algorithm

```
1: $S \leftarrow S_{Off}$                                   ▷ Copy offline schedule
2: repeat
3:     if $\exists s \in S.\ x \in s(I)$ then                ▷ $x$: current location
4:         $v \leftarrow s(v)$
5:         execute $s(job)$
6:         if $x =$ end of $s(I)$ then
7:             $S \leftarrow S \backslash s$
8:         end if
9:     else
10:        $v \leftarrow v_{max}$
11:        Execute any available jobs
12:    end if
13:    Eliminate all $s \in S$ s.t. $s(job)$ is finished
14:    if $\exists s \in S.\ x \in s(I)$ then
15:        $dest \leftarrow$ end of $s(I)$
16:    else if $S \neq \emptyset$ then
17:        $dest \leftarrow$ start of $next(S)$     ▷ Start of the next schedule entry in $S$
18:    else
19:        $dest \leftarrow$ BS                              ▷ BS: base station
20:    end if
21: until arrive at the BS
```

been collected beforehand by opportunistic data collection. Then, the schedule entries covering A-B-C are removed and the next schedule entry to execute is the one covering D-E. So the data mule sets its destination to D and takes a shortcut path P-D. On P-D, the data mule moves at the maximum speed and does opportunistic data collection. Since the length of P-D is shorter than that of P-B-D and the speed is maximum, we can further reduce the travel time.

The following theorem guarantees that the travel time is shorter than that of the offline algorithm:

Theorem 9.2.2. $T_S \leq T_{Off}$

Proof. Let $\mathcal{I}_{idle}$,$\mathcal{I}_{busy}$ denote the sets of location intervals that the data mule is idle and busy in the offline schedule S_{Off}, respectively.

Consider another offline schedule S'_{Off} that is exactly the same in all of $\mathcal{I}_{busy}$ but takes a shortcut between the intervals in $\mathcal{I}_{busy}$, where the data mule moves at the maximum speed. Let T'_{Off} denote the travel time for S'_{Off}, then $T'_{Off} \leq T_{Off}$ clearly holds.

In the semi-online algorithm, when the data mule does not fully cover one of the intervals in $\mathcal{I}_{busy}$, it means that the job has finished before reaching the finishing location as planned in S_{Off}. Since the data mule can directly head to the start of the next schedule entry, it may further reduce[2] the total travel length from S'_{Off}. As movements outside of $\mathcal{I}_{busy}$ are always at the maximum speed, we have $T_S \leq T'_{Off}$ and the theorem follows. □

9.2.4 Numerical Experiments

We implement the 1-D and 2-D semi-online algorithms and the offline algorithm presented in Chapter 4 and Chapter 6 in MATLAB with YALMIP interface [Löf04] and GLPK [GLP] for LP solver.

Fifty nodes are placed randomly in a circular field with the radius of 200m (dense) and 500m (sparse). Base station is located at the center of the circle, and the data mule starts from and comes back to the base station. Each node has the equal amount of data that takes time e sec to be sent to the data mule. We set e to 10 or 30 secs. For the numerical experiments, unknown communication range is assumed to be circular (radius r_U), as well as known communication range. Note that this is just for the purpose of simplifying the experiments and all the algorithms work with the unknown communication range with arbitrary shape. We fix r_U to 150m and changed the ratio r_K/r_U from 0.0 to 1.0 by 0.1 step. This ratio represents the amount of offline knowledge we have in advance. The maximum speed of the data mule is 2 m/sec.

Figure 9.3 shows an example of the path the data mule takes in each of the algorithms.

Figure 9.4 shows the results for different density and execution time. Each graph shows the travel time of data mule for different r_K/r_U. In all cases, travel time decreased as r_K/r_U increased, i.e., when more offline knowledge was available. The change was the largest in the offline algorithm, whereas it was slight in the 2-D semi-online algorithm.

[2]Travel length is unchanged when the interval, the new destination, and the current location are aligned on a line.

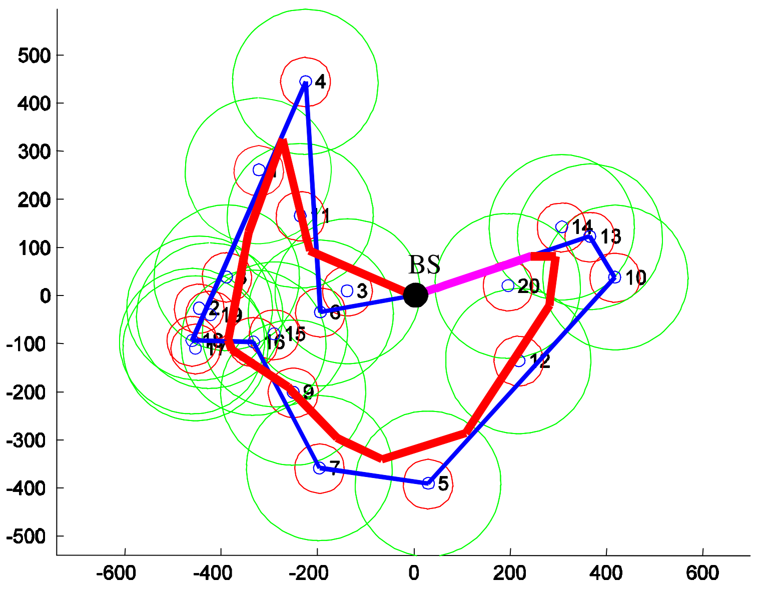


Figure 9.3: Example of data mule's path: 20 nodes, $r_K = 50$m, and $r_U = 150$m. The polygon with thin lines is the offline path that the offline and the 1-D semi-online algorithms choose. The path by the 2-D semi-online algorithm is shown in bold lines.

As Theorems 1 and 2 state, travel time was always shorter in the 1-D and 2-D semi-online algorithms than the offline algorithm.

From these results, we can infer the effects of various assumptions about connectivity in the previous studies. For example, the assumption that no remote communication is possible (as in [SRS07, XWJL08]) corresponds to the case where $r_K/r_U = 0$ in the offline algorithm. This case is up to three times worse than the 2-D semi-online algorithm (in dense, $e = 10$). Another example is the adaptive case proposed in [SKJ$^+$06]. It roughly corresponds to the case where $r_K/r_U = 0$ in the 1-D semi-online algorithm, which is up to 1.8 times as bad as the 2-D semi-online algorithm (in dense, $e = 10$). The fixed-range model (as in [MY07, ZAZ04]) corresponds to the cases where $r_K/r_U = 1$. Although we do not see much difference in travel time between algorithms in this case, it is not likely that we will have the complete knowledge about the communication range in reality.

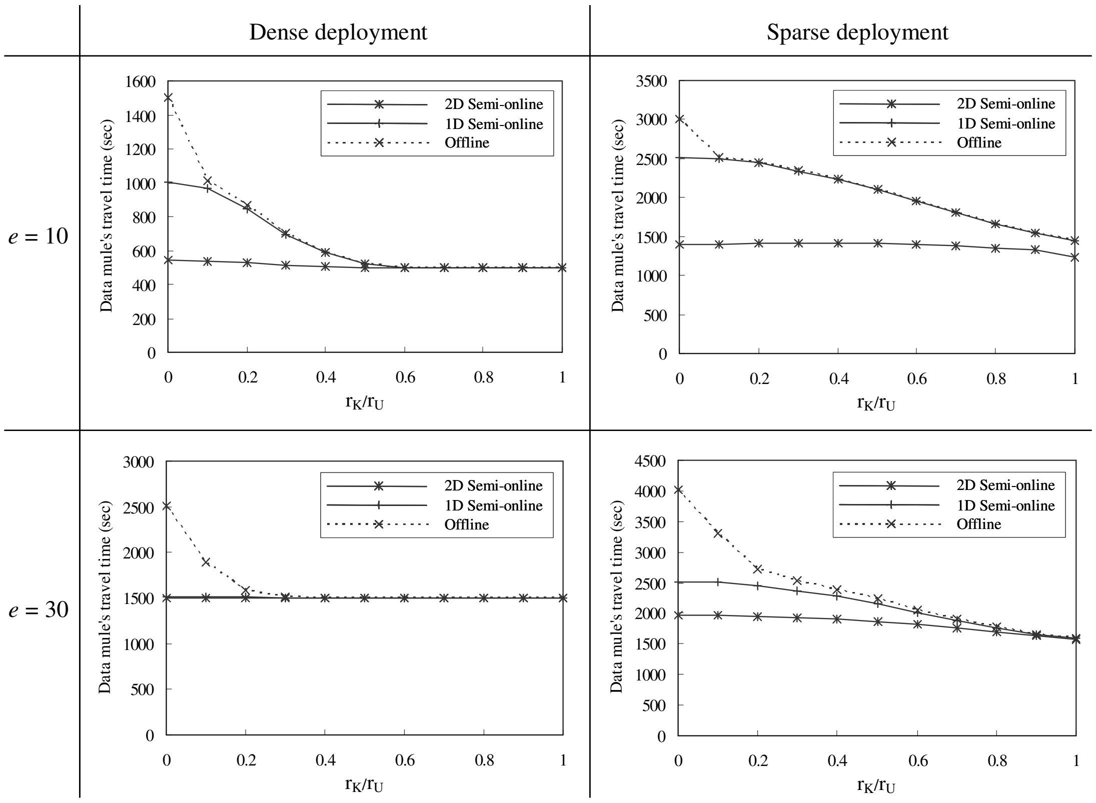


Figure 9.4: Results of numerical experiments: 50 nodes, r_U is fixed to 150m and r_K is varied from 0 to 150m. Travel time is lower-bounded by the total of the execution time of all 50 nodes, which is 500 sec ($e = 10$ case) and 1500 sec ($e = 30$ case).

9.3 Semi-Online Scheduling: Periodic Case

We can use the algorithm for the non-periodic case to design the one for the periodic case. The main change from the non-periodic case will be in deciding whether to skip a schedule entry or not, because the data is continuously generated and a job can never be finished. Our idea is to use a simple strategy: the data mule simply tries to collect from each node as much data as possible according to the offline schedule: i.e., replace "$s(job)$ is finished" in Line 13 of Algorithm 5 with "node $s(job)$ is empty." When a node's buffer once becomes empty, the node is regarded as "finished" and the schedule entries for that node are skipped throughout the current period.

We can construct a periodic offline algorithm in the following way. We use the same path selection algorithm described in Figure 6.2 and solve the 1-D DMS problem for the periodic case. When there is no acceleration constraint, the periodic 1-D DMS problem is solved optimally either by a closed formula or by linear programming, as presented in Section 4.5.

The following two lemmas state that the periodic semi-online algorithm collects more data in shorter amount of time compared to the offline algorithm.

Lemma 9.3.1. *Let $T_{Sp}^{(k)}, T_{Off}$ denote the travel time in the k-th period for the periodic semi-online algorithm and the periodic offline algorithm. Then, for all k, $T_{Sp}^{(k)} \leq T_{Off}$.*

Proof. Immediately follows from Theorem 9.2.2. □

Lemma 9.3.2. *Let $c_i^{(k)}, c_i^{Off}$ denote the amount of data collected from i-th node in the periodic semi-online algorithm (in k-th period) and in the periodic offline algorithm, respectively. Then, for any k and all i, either $c_i^{(k)} \geq c_i^{Off}$ or i-th node's buffer becomes empty during k-th period.*

Proof. If i-th node does not become empty during k-th period, all schedule entries for i-th node are fully executed by the semi-online algorithm. Thus, at least c_i^{Off} is collected. In addition, the semi-online algorithm may collect more data opportunistically. □

Then the following theorem guarantees that the system is stable; i.e., the amount of data in each node does not increase indefinitely:

Theorem 9.3.3. *Assume there exists a feasible offline schedule for a given set of nodes, data generation rate λ_i, stop time T_b, and v_{max}. After sufficiently large number of periods, for the periodic semi-online algorithm, the amount of data in each node is less than some constant.*

Proof. For sufficiently large k, we show that the amount of data in each node is not increasing. We consider the following three cases, classified by the travel time and the amount of collected data: (i) $T_{Sp}^{(k)} = T_{Off}$ and $c_i^{(k)} = c_i^{Off}$ for all i, (ii) $T_{Sp}^{(k)} = T_{Off}$ and $\exists i. c_i^{(k)} \neq c_i^{Off}$, and (iii) $T_{Sp}^{(k)} < T_{Off}$. From Lemma 9.3.1, these three cases enumerate all possibilities. In Case (i), for any node, the amount of data generated and collected are the same. Thus the amount of data in each node is not increasing. In Case (ii), for i's that satisfy $c_i^{(k)} \neq c_i^{Off}$, by Lemma 9.3.2, either $c_i^{(k)} > c_i^{Off}$ or i-th node becomes empty during k-th period. For the first case, the amount of data in the node will decrease. For the second case, the amount is less than c_i^{Off}, which is a constant. Finally in Case (iii), the theorem holds for the nodes that become empty for the same reason as above.

For other nodes, from Lemma 9.3.2, the data mule collects more than c_i^{Off} in the time strictly less than T_{Off}. Thus the amount of data in each node will decrease. □

9.4 Communication Protocol

Let us consider how the data mule communicates with each node in either scheduled or opportunistic data collection. We design a simple request-response-based communication protocol. In this protocol, communications are always initiated by the data mule. This helps keep the implementation simple at the nodes, which have only limited computational resources.

Note that the protocol design described here is one of the simplest examples. We can possibly improve the throughput in several ways, for example by letting nodes send multiple packets per single request and introducing a windowing scheme as in TCP.

9.4.1 Basic Operation

Data from nodes is sent and acknowledged packet by packet in the following way. The data mule sends a request packet that includes a request ID and requested data size. The data mule keeps track of the latest request ID for each node. When a node receives a request, it responds to the data mule by sending the data of the requested size with attaching the request ID. When the data mule receives a packet with the newest request ID for that node, it increments the request ID and sends the next request. If the data mule does not receive a response for a request within the predetermined timeout period, it regards the packet was lost and sends the previous request again.

9.4.2 Scheduled Data Collection

For the scheduled data collections in the offline plan, the data mule sends a request only to the node designated in each schedule entry. It continues to send requests until one of the following events happens:

1. Amount of collected data reached the size designated in the schedule entry.

2. The data mule arrived at the endpoint of the schedule entry.

3. The buffer of the node became empty.

Ideally the first and second events happen at the same time, but they usually do not, due to the error in estimating the effective bandwidth.

9.4.3 Opportunistic Data Collection

For the opportunistic data collection, a practical issue is that the data mule cannot tell whether it is in the unknown communication range of a node without actually communicating with it. This can be accomplished by using advertisement packets. When the data mule tries to start opportunistic data collection, it first broadcasts an advertise packet. When a node receives an advertise packet, it responds to that by sending the size of data in its buffer to the data mule. Then, when the data mule receives the response packet, it sends a request to that node just as in the scheduled data collection. In this way, the data mule can communicate with multiple available nodes concurrently and can adapt dynamically to transient connectivity. To find new nodes in range, the data mule issues advertisement packets periodically during opportunistic data collection.

9.5 Simulation Experiments

To evaluate the effect of introducing the hybrid connectivity model and the benefit of using semi-online algorithms in realistic radio environments, we conduct simulation experiments in ns2 with the Shadowing propagation model.

9.5.1 Methods

We have implemented the offline algorithm and the semi-online algorithm for the DMS problem in the periodic data generation case. We implemented the offline algorithm in MATLAB and generated a Tcl script for ns2. The semi-online algorithm along with the communication protocol are implemented as the modules of ns2. We run the script on ns2 version 2.33.

To assess the performance, we measure the delivery latency for each data packet from the time it is generated until the time the base station receives it. For each test case, the simulation is repeated multiple periods until it reaches stability. We consider it stable when the average delivery latency of the data received in the current period is within $\pm 1\%$ of that of the previous period. If it is stable, we use the data for the next period as the final results.

We use the Shadowing propagation model in ns2. In the Shadowing model, the received power $P_r(d)$ at distance d is derived as the ratio to that at reference distance d_0 as follows:

$$\left[\frac{P_r(d)}{P_r(d_0)}\right]_{dB} = -10\beta \log\left(\frac{d}{d_0}\right) + X_{dB}$$

where β is pass loss exponent, X_{dB} is a Gaussian random variable with zero mean and standard deviation σ_{dB}, which is called the shadowing deviation. Based on [SCA05], we set $d_0 = 1.0, (\beta, \sigma_{dB}) = (3.0, 6.0)$ to simulate outdoor environments. We set the size of known communication range r_K to $20[m]$, where the theoretical successful reception probability is 99.9%.

Other parameters are as follows. Fifty nodes are randomly placed in a circular area of radius $200[m]$ (dense) or $500[m]$ (sparse). We generate 10 node deployments for each. For all deployments, the base station is placed at the center of the circular area and the data mule starts from and comes back to the base station. Data generation rate λ at each node is 100 or $500[Bytes/sec]$. In the communication protocol, the request timeout is 200 msec and the period to issue advertisement packet is 5 sec. In ns2, we use 802.11 MAC with RTS/CTS and bandwidth 2 Mbps. Packet size is set to 400 Bytes. We determined the effective bandwidth by a simple experiment: the data mule and a node are placed 10 m apart and, using the communication protocol above, the data mule tries to collect data as much as possible within 10 sec. The average of 10 measurements with different seeds for random number generator of ns2 was 402440 Bytes, which corresponds to 322.0 Kbps. Based on these results, we use 320 Kbps as the effective bandwidth.

9.5.2 Main Results

Figure 9.5 shows the average data delivery latency for each of the four deployment cases. All tested cases reached the stability condition. Average number of periods until getting stable was 4.0 (min: 4, max: 4) for the offline algorithm and 5.3 (min: 4, max: 8) for the semi-online algorithm, respectively. The average data delivery latency was lower in the semi-online algorithm in all cases. The decrease was larger in the dense deployments (38.9% and 20.6% for λ =100 and 500, respectively) than in the sparse deployments (15.4% and 17.4% for λ =100 and 500, respectively).

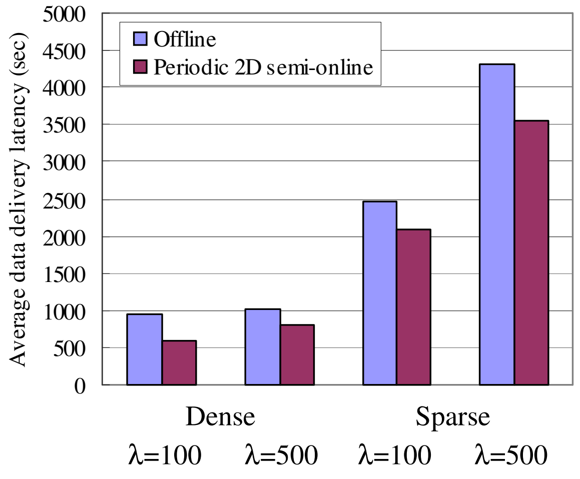


Figure 9.5: Simulation results: Average data delivery latency: 50 nodes, average of 10 experiments for each case.

9.5.3 Effects of Inaccurate Parameters

For the hybrid connectivity model and the algorithms (both offline and semi-online) to work, we need to estimate two parameters: the size of known communication range (r_K) and the effective bandwidth. Larger r_K implies better performance, but it is not clear about the consequences when it is larger than the reality. We have a similar issue for effective bandwidth, too, especially when the actual effective bandwidth fluctuates over time. Here we see the effects of overestimating these parameters on both the offline and the semi-online algorithms.

We use "collection rate" as the performance metric for these experiments. Collection rate R_C is calculated in each period and, for k-th period, it is defined as follows:

$$R_C^{(k)} = \frac{\sum_i c_i^{(k)}}{\sum_i g_i^{(k)}},$$

where $g_i^{(k)}$ is the amount of data generated at i-th node in k-th period and $c_i^{(k)}$ is the amount of data collected by the data mule in k-th period. For the data to be collected without any loss, R_C needs to be 1 on average. On the other hand, if R_C is constantly lower than 1, data accumulates at each node over time and eventually overflows, resulting in loss of data.

As in the previous experiments, we test each case on four deployments: combinations of two node densities (dense and sparse) and two data generation rates ($\lambda = 100$

and 500). For the experiments on known communication range, we test on $r_k = 20$ (default), 30, 40, and 50. Theoretical probabilities of successful reception for these values are 99.9%, 98.4%, 93.6%, and 85.1%, respectively. For the experiments on effective bandwidth, we test on 320 Kbps (default), 480 Kbps, and 640 Kbps.

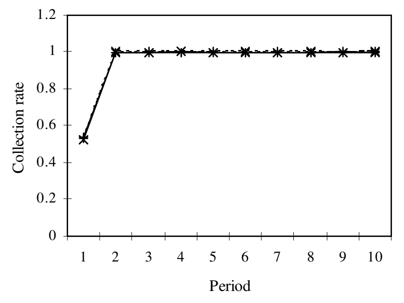
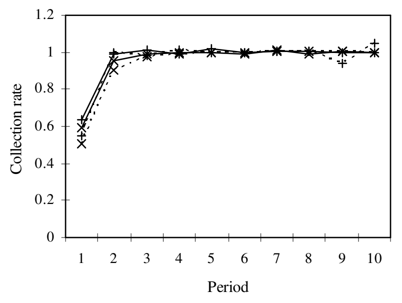
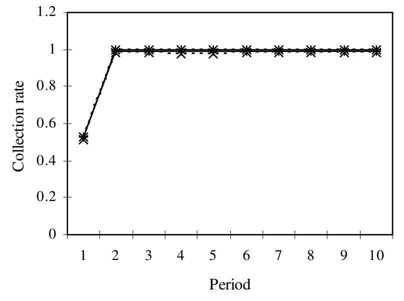
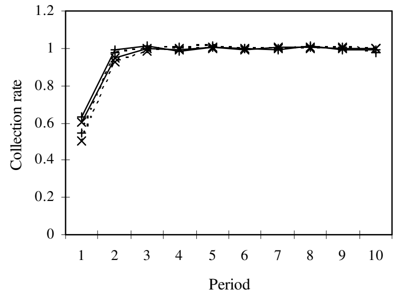
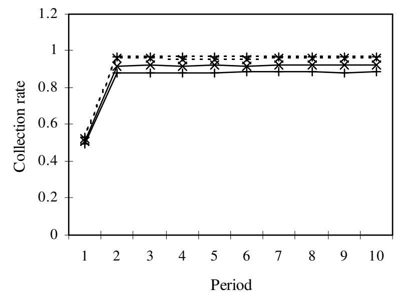
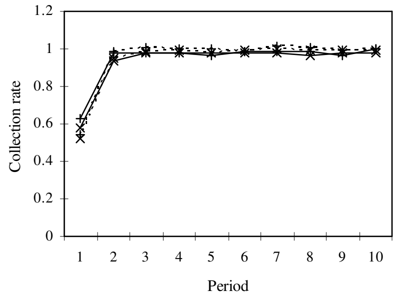
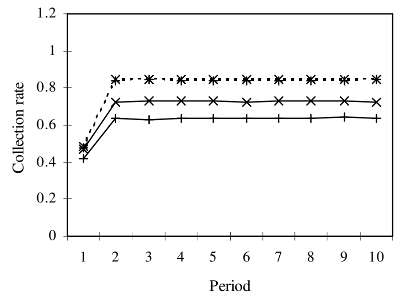
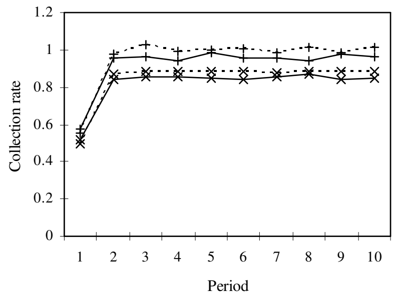
Figure 9.6 shows the effect of overestimating r_K. In the offline algorithm, the average of R_C for the deployments (from period 6 to 10) was 1.000, 0.991, 0.933, 0.762 for r_K =20, 30, 40, 50, respectively. The average was higher in the semi-online algorithm: 1.000, 0.999, 0.989, and 0.923, respectively.

Figure 9.7 shows the effect of overestimating the effective bandwidth. The average of R_C for the deployments (from period 6 to 10) was 1.000, 0.679, 0.509 for the 320 Kbps case, 480 Kbps case, 640 Kbps case, respectively. These values almost agree with the ratio to the actual effective bandwidth ($\approx$ 322 Kbps). For the semi-online algorithm, the average of R_C was much higher: 1.000, 0.989, 0.983 for the effective bandwidth of 320 Kbps, 480 Kbps, 640 Kbps, respectively.

To summarize, the experiments with overestimated r_K and effective bandwidth showed that, in these cases, we can achieve higher collection rate in the semi-online algorithm than in the offline algorithm. This suggests that the semi-online algorithm is beneficial in terms of the robustness against inaccurate parameters, as well as the performance improvements as demonstrated in the first experiments.

9.6 Related Work

The term "semi-online scheduling" appears in the context of job scheduling and refers to the cases when partial information is available offline. Common examples of partial information are optimum makespan, order of job arrivals, etc.; for more on these topics, see [PST04]. The notion of "worst case execution time" in real-time scheduling problems is analogous in the sense that only the upper bound of execution time is given offline and actual execution time is known at runtime. Also in real-time scheduling context, a loosely related work is Imprecise computation [Liu00], which is the model in which each task consists of mandatory and optional components. When the system is overloaded, optional components may be skipped, but the results of the task will be "imprecise" or approximate. In our case, partial information is the feasible location intervals. Specifically, part of the feasible location intervals is known in advance, but the job is possibly executable also in other intervals. Unlike work in the real-time systems

	Offline	Semi-online
$r_K = 20m$ (Reception prob.: 99.9%)		
$r_K = 30m$ (Reception prob.: 98.4%)		
$r_K = 40m$ (Reception prob.: 93.6%)		
$r_K = 50m$ (Reception prob.: 85.1%)		

—+— Dense, λ = 100 Bytes/sec	—×— Dense, λ = 500 Bytes/sec
···+··· Sparse, λ = 100 Bytes/sec	···×··· Sparse, λ =500 Bytes/sec

Figure 9.6: Simulation results: Effect of overestimating r_K (size of known communication range).

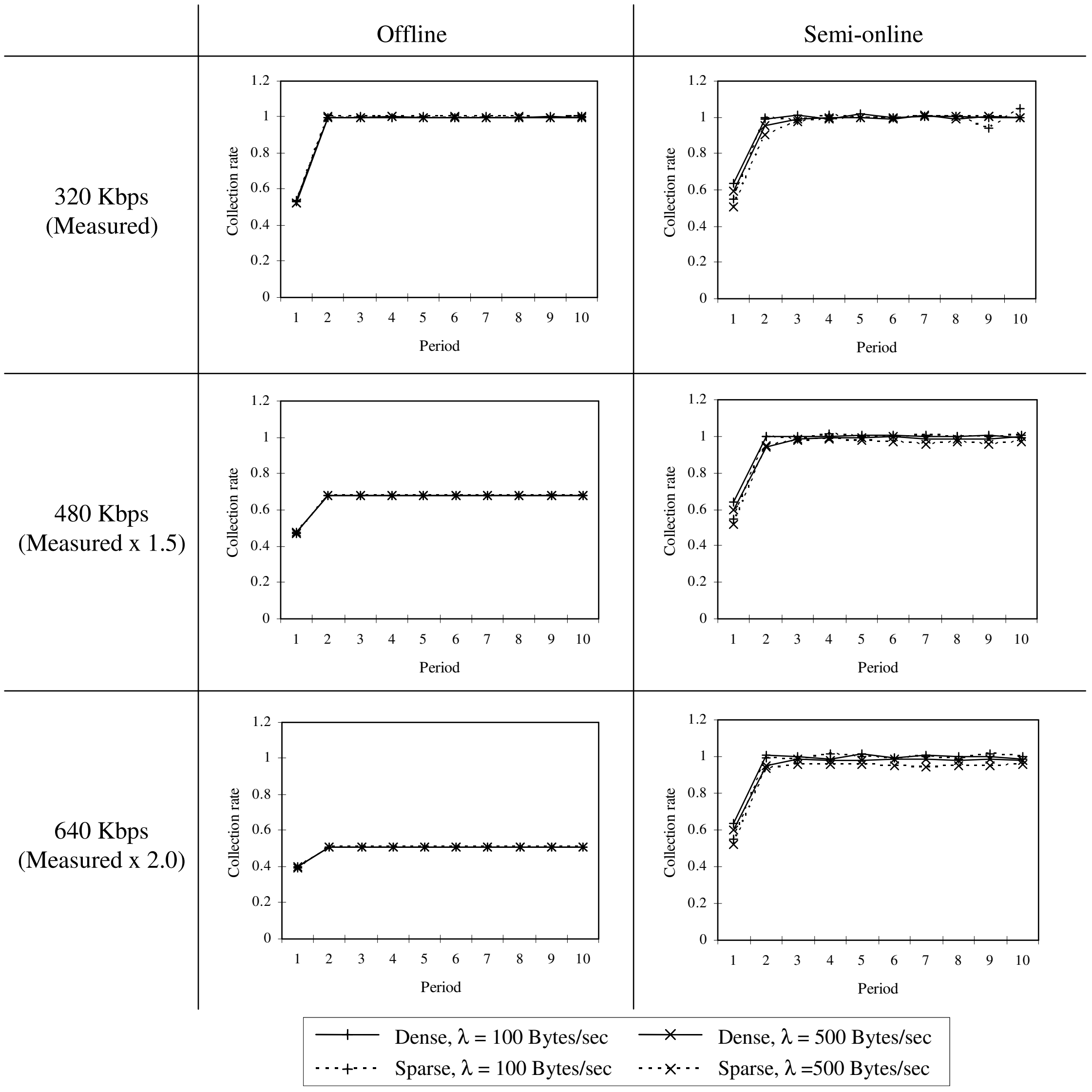


Figure 9.7: Simulation results: Effect of overestimating the effective bandwidth.

community, this is not a standard assumption, since scheduling problems for jobs with multiple feasible intervals are not common.

9.7 Summary

We have formulated the problem of controlling a data mule under a realistic connectivity model, which we call the hybrid model. Then we designed semi-online scheduling algorithms that make an offline plan and update it at runtime. We proved that these algorithms are guaranteed to perform better than the offline algorithm. Numerical experiments showed significant improvements over the offline algorithm, especially when there is less information on the communication range known in advance. Then we extend the semi-online algorithm to periodic data generation case. We also designed a communication protocol that keeps the implementation at the node very simple, which is good for resource-constrained nodes. Finally, in simulation experiments with ns2 under Shadowing model, we demonstrated that the data delivery latency is reduced by up to 38% in the semi-online algorithm compared to the offline algorithm. We also showed that the semi-online algorithm is robust against inaccurate estimations of radio parameters. These results suggest that the semi-online algorithms are feasible in realistic radio environments.

Acknowledgements This chapter, in part, has been submitted for publication as "Path Planning of Data Mules in Sensor Networks" by Ryo Sugihara and Rajesh K. Gupta in ACM Transactions on Sensor Networks [SG09d]. The dissertation author was the primary investigator and author of this paper.

This chapter, in part, has been submitted for publication as "Speed Control and Scheduling of Data Mules in Sensor Networks" by Ryo Sugihara and Rajesh K. Gupta in ACM Transactions on Sensor Networks [SG09e]. The dissertation author was the primary investigator and author of this paper.

Chapter 10

Conclusion and Future Work

Use of controlled mobility in sensor networks enables data mule approach for data collection from sensor nodes. Compared to the widely-used multihop forwarding approach, the data mule approach is beneficial in reducing the energy consumption at the sensor nodes by eliminating the need for forwarding other sensors' data. On the other hand, increased data delivery latency is the most notable disadvantage in the data mule approach.

To reduce the data delivery latency, optimizing the motion of the data mule is critical, since the latency is mostly governed by the relatively slow movement of the data mule compared to the speed of data traveling over wireless links. However, motion optimization is a difficult problem, since we need to choose the speed, path, and data collection schedule simultaneously such that the data mule's travel time is minimized. Due to the hardness, previous literature often simplified the problem so that they can solve the problem optimally and/or design heuristic algorithms. These simplification lead to suboptimal solutions and also made the algorithms difficult to be applied to similar but slightly different application scenarios.

In this dissertation, we have presented a problem framework called the Data Mule Scheduling (DMS) problem for optimization of data mules' motion. The DMS problem framework divides the motion optimization problem into four subproblems (Forwarding, Path selection, Speed control, and Job scheduling) and addresses them separately when necessary. The benefits of the DMS problem framework are summarized by the following four points:

Expressiveness The DMS problem framework is general and capable of expressing

Table 10.1: Summary of complexity results for the 1-D DMS problem

	Simple (location) jobs		General (location) jobs	
	Offline	Online	Offline	Online
Preemptive Job Scheduling (Section 4.2)	EDD algorithm [Jac55, LL73, SSNB95]		LP	Non-existent (Thm. 4.2.1)
Constant Speed 1-D DMS (Section 4.3)	$O(n^2)$ (FIND-MIN-MAXSPEED)	Non-existent	LP	Non-existent
Variable Speed 1-D DMS (Section 4.4)	$O(n^3)$ (SEQUENTIAL-FIND-MIN-MAXSPEED)	[$v_{min} = 0$] EDD-WITH-STOP	LP	Non-existent (Thm. 4.4.3)
		[$v_{min} > 0$] Non-existent (Thm. 4.4.2)		
Generalized 1-D DMS (Chapter 5)	(Open) / PTAS (Thm. 5.3.1)	Non-existent	[fixed $k \geq 2$] NP-hard (Thm. 5.2.1)	Non-existent
			[k arbitrary] Strongly NP-hard (Thm. 5.2.2)	

various different problem settings and assumptions. These variations include fixed path (1-D) case, pure data mule approach, and hybrid data mule approach, and all of which can be combined with different connectivity and mobility models. This expressiveness allows us to handle the several assumptions used in the previous literature, such as constant speed motion of data mule and zero communication ranges, in the DMS problem framework.

Effectiveness In terms of theoretical analysis, the DMS problem framework has enabled us to extract optimally-solvable cases out of the hard problem. It also allowed us to identify the hard cases that have motivated us to design approximate and heuristic algorithms. Table 10 summarizes the complexity results for the 1-D DMS problem discussed in Chapter 4 and Chapter 5.

The DMS framework is also effective for improving the performance as demonstrated by the simulation experiments. For example in Chapter 6, simulation experiments showed that our formulation of the path selection problem as the Label-Covering Tour problem, along with the optimal algorithm for the 1-D DMS problem, has been effective for improving the data delivery latency compared to other techniques proposed in the literature.

Flexibility The DMS problem framework is flexible in the sense that it accommodates different problem formulations other than the ones presented in this dissertation. Ultimately, each subproblem is formulated in an arbitrary manner as long as the interface with neighboring subproblems are maintained. An example of alternative problem formulation is the one for the path selection problem as TSP with Neighborhoods problem [DM03], as discussed in Chapter 6.

Extensibility The DMS problem framework is extensible in several ways to be applied to various application scenarios. The extensibility has been demonstrated in Chapter 8 for multiple data mules case and in Chapter 9 for the case of partially-known communication ranges.

Our work in this dissertation has been focused on designing a versatile problem framework for optimizing the motion of a data mule and providing some basic problem formulations and analyses. Naturally there are many things to be done in the area, as listed below.

Open cases on complexity analysis The computational complexity of the Generalized 1-D DMS problem with simple location jobs remains unknown. Clearly it is easier than the general location jobs case, since simple location jobs are a special case of general location jobs. This is also suggested by the fact that our approximation scheme cannot be used at least in its original form for the general location jobs case, since the feasibility condition cannot be expressed by cumulative execution time. However, as of this writing, it is not clear if the problem is easy enough to be solved optimally in polynomial time.

Relaxing the assumptions We started with a number of simplifying assumptions that enabled us to complete the problem formulation of the entire DMS problem. We later relaxed a few important assumptions. As an example of such efforts, in Chapter 9 we have relaxed the assumption of fixed and known communication range and have designed the semi-online scheduling algorithms. Among the remaining assumptions, the constant bandwidth assumption is one of the strongest. One way to relax this assumption would be to use a conservative estimate for the bandwidth and to treat the execution time as the worst case execution time (WCET). Then, for the case when a data collection job finishes earlier than the WCET, we need an adaptive algorithm that makes use of the extra time for improving the data mule's travel time. The semi-online scheduling algorithms for the case of partially-known communication ranges may realize this in part, but we need further analyses.

Another strong assumption is the complete knowledge about location of both data mule and sensor nodes. There may also be a deviation between planned and actual movements of data mule. As well as the previous case, the semi-online scheduling algorithms are helpful for absorbing these errors to some extent, but we can possibly design a better algorithm to deal with them.

Different problem settings We can think of several different problem settings. An example is to use an objective other than minimizing the data delivery latency. For instance, the problem studied in Somasundara et al. [SRS04, SRS07] was to find a data mule's schedule so that the buffers of sensor nodes do not overflow. Another example is the case of multiple base stations when there are multiple data mules. This problem requires different problem formulation for the subproblems of forwarding and path selection. To handle these cases, we need to modify or extend the DMS problem framework.

Different problem formulations Not only the problem settings, we can also use different problem formulations for the DMS framework other than the ones we have presented. For example, the formulation for the forwarding problem has been based on the simplification where the objective is to minimize the weighted distance from the base station, but it would be better if we can optimize the forwarding strategy in a way that is more directly connected to the data mule's travel time. A possible starting point is to consider the forwarding problem and the path selection problem together so that a better forwarding strategy leads to a shorter path.

Broader application area Although the DMS problem framework is designed primarily for data mule approach in data-collection applications, it is potentially useful also for other applications in sensor networks. For instance, data dissemination is another important aspect of sensor networks when the users need to update a program module in the sensor node or the network contains actuator nodes that can be remotely commanded. Using a data mule for these purposes is feasible and, in fact, it is possible to model these cases within the DMS problem framework by treating both upward and downward communications simply as location jobs.

The DMS framework is also potentially extensible for a mobile ad-hoc network scenario where a data mule is used for mediating node-to-node communications without a base station. Message Ferrying [ZA03, ZAZ05] represents this scenario. Their objective is to minimize the average data delivery delay and meet bandwidth requirement of each node, assuming that the traffic between each pair of nodes is given. They designed some algorithms for single ferry case [ZA03] and multiple ferries case [ZAZ05], but they only considered constant speed mobility model and used zero communication ranges (in [ZAZ05]). By extending the DMS problem framework, we may be able to address the problem in a more detailed and theoretically grounded way. Specifically, for node-to-node communications, a data mule needs to receive data from a source node before sending the data to the destination node. In the context of job scheduling, this is analogous to the case where there are dependencies among the jobs.

Beyond sensor networks An interesting and adventurous future work is to use the DMS problem framework as a proxy for the problems in different domains other than sensor networks. One such example has been shown in the connection between the 1-D DMS problem and the processor speed scaling problem discussed in Chapter 4 and Chapter 5. Another possibility is kinodynamic motion planning in robotics area that we have used for designing the approximation scheme for the Generalized 1-D DMS problem. Conversely applying the algorithms in the DMS problem to motion planning problems in robotics area may be interesting if there is a good application for that.

Bibliography

[AHL06] Esther M. Arkin, Refael Hassin, and Asaf Levin. Approximations for minimum and min-max vehicle routing problems. *Journal of Algorithms*, 59(1):1–18, 2006.

[AMS07] Susanne Albers, Fabian Müller, and Swen Schmelzer. Speed scaling on parallel processors. In *SPAA '07: Proceedings of the 19th annual ACM symposium on Parallel algorithms and architectures*, pages 289–298, 2007.

[AS08] Claudia Archetti and Maria Grazia Speranza. The split delivery vehicle routing problem: A survey. In Bruce Golden, S. Raghavan, and Edward Wasil, editors, *The Vehicle Routing Problem: Latest Advances and New Challenges*. Springer Science+Business Media, New York, NY, 2008.

[Bal89] Egon Balas. The prize collecting traveling salesman problem. *Networks*, 19(6):621–636, 1989.

[BC03] Seema Bandyopadhyay and Edward J. Coyle. An energy efficient hierarchical clustering algorithm for wireless sensor networks. In *INFOCOM '03: Proceedings of the 22th Annual Joint Conference of the IEEE Computer and Communications Societies*, pages 1713–1723, 2003.

[BCK$^+$07] Nikhil Bansal, Ho-Leung Chan, Rohit Khandekar, Kirk Pruhs, Baruch Schieber, and Cliff Stein. Non-preemptive min-sum scheduling with resource augmentation. In *FOCS '07: Proceedings of the 48th IEEE Symposium on Foundations of Computer Science*, pages 614–624, 2007.

[BHR93] Sanjoy K. Baruah, Rodney R. Howell, and Louis E. Rosier. Feasibility problems for recurring tasks on one processor. *Theoretical Computer Science*, 118(1):3–20, 1993.

[BKP07] Nikhil Bansal, Tracy Kimbrel, and Kirk Pruhs. Speed scaling to manage energy and temperature. *Journal of the ACM*, 54(1):1–39, 2007.

[BRY$^+$04] Maxim A. Batalin, Mohammad Rahimi, Yan Yu, Duo Liu, Aman Kansal, Gaurav S. Sukhatme, William J. Kaiser, Mark Hansen, Gregory J. Pottie, Mani Srivastava, and Deborah Estrin. Call and response: experiments in sampling the environment. In *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 25–38, 2004.

[BV04] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, New York, NY, USA, 2004.

[CHZ02] Maurice Chu, Horst Haussecker, and Feng Zhao. Scalable information-driven sensor querying and routing for ad hoc heterogeneous sensor networks. *International Journal of High Performance Computing Applications*, 16(3):90–110, 2002.

[Con] Concorde TSP Solver. `http://www.tsp.gatech.edu/concorde/index.html`.

[CRM$^+$08] Kameswari Chebrolu, Bhaskaran Raman, Nilesh Mishra, Phani Kumar Valiveti, and Raj Kumar. BriMon: A sensor network system for railway bridge monitoring. In *MobiSys '08: Proceedings of the 6th international conference on Mobile systems, applications, and services*, pages 2–14, 2008.

[CSA03] Arnab Chakrabarti, Ashutosh Sabharwal, and Behnaam Aazhang. Using predictable observer mobility for power efficient design of sensor networks. In *IPSN '03: Proceedings of the 2nd international symposium on Information processing in sensor networks*, pages 129–145, 2003.

[CWSK05] Jian-Jia Chen, Jun Wu, Chi-sheng Shih, and Tei-Wei Kuo. Approximation algorithms for scheduling multiple feasible interval jobs. In *RTCSA '05: Proceedings of the 11th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 11–16, 2005.

[DM03] Adrian Dumitrescu and Joseph S. B. Mitchell. Approximation algorithms for TSP with neighborhoods in the plane. *Journal of Algorithms*, 48(1):135–159, 2003.

[DXCR93] Bruce Donald, Patrick Xavier, John Canny, and John Reif. Kinodynamic motion planning. *Journal of the ACM*, 40(5):1048–1066, 1993.

[EFMS05] Khaled M. Elbassioni, Aleksei V. Fishkin, Nabil H. Mustafa, and René Sitters. Approximation algorithms for euclidean group tsp. In *ICALP '05: Proceedings of the 32nd International Colloquium on Automata, Languages and Programming*, pages 1115–1126, 2005.

[EGB06] Eylem Ekici, Yaoyao Gu, and Doruk Bozdag. Mobility-based communication in wireless sensor networks. *IEEE Communications Magazine*, 44(7):56–62, July 2006.

[ER08] Friedrich Eisenbrand and Thomas Rothvoß. A PTAS for static priority real-time scheduling with resource augmentation. In *ICALP '08: Proceedings of the 35th International Colloquium on Automata, Languages and Programming*, pages 246–257, 2008.

[FHK78] Greg N. Frederickson, Matthew S. Hecht, and Chul E. Kim. Approximation algorithms for some routing problems. *SIAM Journal on Computing*, 7(2):178–193, 1978.

[GBBE06] Yaoyao Gu, Doruk Bozdağ, Robert W. Brewer, and Eylem Ekici. Data harvesting with mobile elements in wireless sensor networks. *Computer Networks*, 50(17):3449–3465, 2006.

[GDPV03] Shashidhar Rao Gandham, Milind Dawande, Ravi Prakash, and S. Venkatesan. Energy efficient schemes for wireless sensor networks with multiple mobile base stations. In *GLOBECOM '03: Proceedings of the IEEE Global Telecommunications Conference*, pages 377–381, 2003.

[GJ79] Michael R. Garey and David S. Johnson. *Computers and intractability: A guide to the theory of NP-completeness.* Freeman, 1979.

[GK00] Piyush Gupta and P. R. Kumar. The capacity of wireless networks. *IEEE Transactions on Information Theory*, 46(2):388–404, 2000.

[GKW$^+$02] Deepak Ganesan, Bhaskar Krishnamachari, Alec Woo, David Culler, Deborah Estrin, and Stephen Wicker. Complex behavior at scale: An experimental study of low-power wireless sensor networks. *UCLA Computer Science Technical Report*, UCLA/CSD-TR 02-0013, 2002.

[GLP] GNU linear programming kit. `http://www.gnu.org/software/glpk/glpk.html`.

[GLS97] Michel Gendreau, Gilbert Laporte, and Frédéric Semet. The covering tour problem. *Operations Research*, 45(4):568–576, 1997.

[Gra69] Ronald L. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics*, 17(2):416–429, 1969.

[GT02] Matthias Grossglauser and David N. C. Tse. Mobility increases the capacity of ad hoc wireless networks. *IEEE/ACM Transactions on Networking*, 10(4):477–486, 2002.

[HCB00] Wendi R. Heinzelman, Anantha Chandrakasan, and Hari Balakrishnan. Energy-efficient communication protocol for wireless microsensor networks. In *Proceedings of the 33rd Annual Hawaii International Conference on System Sciences*, 2000.

[HKS$^+$04] Tian He, Sudha Krishnamurthy, John A. Stankovic, Tarek Abdelzaher, Liqian Luo, Radu Stoleru, Ting Yan, Lin Gu, Jonathan Hui, and Bruce Krogh. Energy-efficient surveillance system using wireless sensor networks. In *MobiSys'04: Proceedings of the 2nd international conference on Mobile systems, applications, and services*, pages 270–283, 2004.

[HQPS98] Inki Hong, Gang Qu, Miodrag Potkonjak, and Mani B. Srivastava. Synthesis techniques for low-power hard real-time systems on variable voltage processors. In *RTSS '98: Proceedings of the 19th IEEE international Real-Time Systems Symposium*, pages 178–187, 1998.

[HY05] John Heidemann and Wei Ye. Energy conservation in sensor networks at the link and network layers. In Nirupama Bulusu and Sanjay Jha, editors, *Wireless Sensor Networks*. Artech House, 2005.

[IGE00] Chalermek Intanagonwiwat, Ramesh Govindan, and Deborah Estrin. Directed diffusion: a scalable and robust communication paradigm for sensor networks. In *MobiCom'00: Proceedings of the 6th annual international conference on Mobile computing and networking*, pages 56–67, 2000.

[ISG03] Sandy Irani, Sandeep Shukla, and Rajesh Gupta. Algorithms for power savings. In *SODA '03: Proceedings of the 14th annual ACM-SIAM symposium on Discrete algorithms*, pages 37–46, 2003.

[IY98] Tohru Ishihara and Hiroto Yasuura. Voltage scheduling problem for dynamically variable voltage processors. In *ISLPED '98: Proceedings of the 1998 international symposium on Low power electronics and design*, pages 197–202, 1998.

[Jac55] James R. Jackson. Scheduling a production line to minimize maximum tardiness. *Research report 43, Management Science Research Project, University of California, Los Angeles*, 1955.

[JOW$^+$02] Philo Juang, Hidekazu Oki, Yong Wang, Margaret Martonosi, Li Shiuan Peh, and Daniel Rubenstein. Energy-efficient computing for wildlife tracking: design tradeoffs and early experiences with ZebraNet. In *ASPLOS-X: Proceedings of the 10th international conference on Architectural support for programming languages and operating systems*, pages 96–107, 2002.

[JSS05] David Jea, Arun A. Somasundara, and Mani B. Srivastava. Multiple controlled mobile elements (data mules) for data collection in sensor networks. In *DCOSS '05: Proceedings of the 1st international conference on Distributed Computing in sensor systems*, pages 244–257, 2005.

[Kar78] Richard M. Karp. A characterization of the minimum cycle mean in a digraph. *Discrete Mathematics*, 23:309–311, 1978.

[KNE03] David Kotz, Calvin Newport, and Chip Elliott. The mistaken axioms of wireless-network research. *Dartmouth College Computer Science Technical Report*, TR2003-467, 2003.

[KSJ$^+$04] Aman Kansal, Arun A. Somasundara, David D. Jea, Mani B. Srivastava, and Deborah Estrin. Intelligent fluid infrastructure for embedded networks. In *MobiSys '04: Proceedings of the 2nd international conference on Mobile systems, applications, and services*, pages 111–124, 2004.

[LCD03] Jessica D. Lundquist, Daniel R. Cayan, and Michael D. Dettinger. Meteorology and hydrology in Yosemite national park: A sensor network application. In *IPSN'03: Proceedings of the 2nd international symposium on Information processing in sensor networks*, pages 518–528, 2003.

[LCL07] HyungJune Lee, Alberto Cerpa, and Philip Levis. Improving wireless simulation through noise modeling. In *IPSN '07: Proceedings of the 6th international conference on Information processing in sensor networks*, pages 21–30, 2007.

[LH05] Jun Luo and Jean-Pierre Hubaux. Joint mobility and routing for lifetime elongation in wireless sensor networks. In *INFOCOM '05: Proceedings of the 24th Annual Joint Conference of the IEEE Computer and Communications Societies*, pages 1735–1746, 2005.

[Liu00] Jane W. S. Liu. *Real-time systems*. Prentice Hall, 2000.

[LL73] C. L. Liu and James W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 20(1):46–61, 1973.

[Löf04] Johan Löfberg. Yalmip : A toolbox for modeling and optimization in MATLAB. In *Proceedings of the CACSD Conference*, Taipei, Taiwan, 2004.

[LR02] Stephanie Lindsey and Cauligi S. Raghavendra. PEGASIS: Power-efficient gathering in sensor information systems. In *Proceedings of the IEEE Aerospace Conference Proceedings*, volume 3, pages 1125–1130, 2002.

[MFA07] Guoqiang Mao, Bariş Fidan, and Brian D. O. Anderson. Wireless sensor network localization techniques. *Computer Networks*, 51(10, 11):2529–2553, 2007.

[MFL$^+$08] David Mascarenas, Eric Flynn, Kaisen Lin, Kevin Farinholt, Gyuhae Park, Rajesh Gupta, Michael Todd, and Charles Farrar. Demonstration of a roving-host wireless sensor network for rapid assessment monitoring of structural health. volume SPIE 6933, page 69330K. SPIE, 2008.

[MHO04] Kirk Martinez, Jane K. Hart, and Royan Ong. Environmental sensor networks. *IEEE Computer*, 37(8):50–56, 2004.

[MY06] Ming Ma and Yuanyuan Yang. SenCar: An energy efficient data gathering mechanism for large scale multihop sensor networks. In *DCOSS '06: Proceedings of the 2nd international conference on Distributed Computing in sensor systems*, pages 498–513, 2006.

[MY07] Ming Ma and Yuanyuan Yang. SenCar: An energy efficient data gathering mechanism for large-scale multihop sensor networks. *IEEE Transactions on Parallel and Distributed System*, 18(10):1476–1488, 2007.

[ns2] ns2 network simulator. `http://www.isi.edu/nsnam/ns/`.

[NYM97] Won Namgoong, Mengchen Yu, and Teresa Meng. A high-efficiency variable-voltage CMOS dynamic DC-DC switching regulator. In *ISSCC '97: Proceedings of the 43rd IEEE International Solid-State Circuits Conference*, pages 380–381, 1997.

[Pat03] Gábor Pataki. Teaching integer programming formulations using the traveling salesman problem. *SIAM Review*, 45(1):116–123, 2003.

[PB94] Charles E. Perkins and Pravin Bhagwat. Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers. In *SIGCOMM '94: Proceedings of the conference on Communications architectures, protocols and applications*, pages 234–244, 1994.

[PPK03] Sundeep Pattem, Sameera Poduri, and Bhaskar Krishnamachari. Energy-quality tradeoffs for target tracking in wireless sensor networks. In *IPSN'03: Proceedings of the 2nd international symposium on Information processing in sensor networks*, pages 32–46, 2003.

[PS01] Padmanabhan Pillai and Kang G. Shin. Real-time dynamic voltage scaling for low-power embedded operating systems. In *SOSP '01: Proceedings of the eighteenth ACM symposium on Operating systems principles*, pages 89–102, 2001.

[PST04] Kirk Pruhs, Jiři Sgall, and Eric Torng. Online scheduling. In Joseph Y-T. Leung, editor, *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. CRC Press, 2004.

[SCA05] Patrick Stuedi, Oscar Chinellato, and Gustavo Alonso. Connectivity in the presence of shadowing in 802.11 ad hoc networks. In *WCNC'05: Proceedings of IEEE Wireless Communications and Networking Conference*, volume 4, pages 2225–2230, 2005.

[SG08a] Ryo Sugihara and Rajesh K. Gupta. Improving the data delivery latency in sensor networks with controlled mobility. In *DCOSS '08: Proceedings of the 4th international conference on Distributed Computing in sensor systems*, 2008.

[SG08b] Ryo Sugihara and Rajesh K. Gupta. Programming models for sensor networks: A survey. *ACM Transactions on Sensor Networks*, 4(2):1–29, 2008.

[SG09a] Ryo Sugihara and Rajesh K. Gupta. Complexity of motion planning of data mule for data collection in wireless sensor networks. *Submitting*, 2009.

[SG09b] Ryo Sugihara and Rajesh K. Gupta. Optimal speed control of mobile node for data collection in sensor networks. *IEEE Transactions on Mobile Computing, to appear*, 2009.

[SG09c] Ryo Sugihara and Rajesh K. Gupta. Optimizing energy-latency tradeoff in sensor networks with controlled mobility. In *INFOCOM '09 (Mini-conference): Proceedings of the 28th Annual Joint Conference of the IEEE Computer and Communications Societies*, pages 2566–2570, 2009.

[SG09d] Ryo Sugihara and Rajesh K. Gupta. Path planning of data mules in sensor networks. *Submitting*, 2009.

[SG09e] Ryo Sugihara and Rajesh K. Gupta. Speed control and scheduling of data mules in sensor networks. *Submitting*, 2009.

[SH03] Tara Small and Zygmunt J. Haas. The shared wireless infostation model: a new ad hoc networking paradigm (or where there is a whale, there is a way). In *MobiHoc '03: Proceedings of the 4th ACM international symposium on Mobile ad hoc networking and computing*, pages 233–244, 2003.

[SKJ+06] Arun A. Somasundara, Aman Kansal, David D. Jea, Deborah Estrin, and Mani B. Srivastava. Controllably mobile infrastructure for low energy embedded networks. *IEEE Transactions on Mobile Computing*, 5(8):958–973, 2006.

[SLC03] Chi-sheng Shih, Jane W. S. Liu, and Infan Kuok Cheong. Scheduling jobs with multiple feasible intervals. In *RTCSA '03: Proceedings of the 9th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 53–71, 2003.

[SML+04] Gyula Simon, Miklós Maróti, Ákos Lédeczi, György Balogh, Branislav Kusy, András Nádas, Gábor Pap, János Sallai, and Ken Frampton. Sensor network-based countersniper system. In *SenSys'04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 1–12, 2004.

[SOP+04] Robert Szewczyk, Eric Osterweil, Joseph Polastre, Michael Hamilton, Alan Mainwaring, and Deborah Estrin. Habitat monitoring with sensor networks. *Communications of the ACM*, 47(6):34–40, 2004.

[SRJB03] Rahul C. Shah, Sumit Roy, Sushant Jain, and Waylon Brunette. Data MULEs: modeling a three-tier architecture for sparse sensor networks. In *Proceedings of the First IEEE International Workshop on Sensor Network Protocols and Applications*, pages 30–41, 2003.

[SRS04] Arun A. Somasundara, Aditya Ramamoorthy, and Mani B. Srivastava. Mobile element scheduling for efficient data collection in wireless sensor networks with dynamic deadlines. In *RTSS '04: Proceedings of the 25th IEEE international Real-Time Systems Symposium*, pages 296–305, 2004.

[SRS07] Arun A. Somasundara, Aditya Ramamoorthy, and Mani B. Srivastava. Mobile element scheduling with dynamic deadlines. *IEEE Transactions on Mobile Computing*, 6(4):395–410, 2007.

[SS84] Barbara Simons and Michael Sipser. On scheduling unit-length jobs with multiple release time/deadline intervals. *Operations Research*, 32(1):80–88, 1984.

[SSNB95] John A. Stankovic, Marco Spuri, Marco Di Natale, and Giorgio C. Buttazzo. Implications of classical scheduling results for real-time systems. *Computer*, 28(6):16–25, Jun 1995.

[Stu99] Jos F. Sturm. Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones. *Optimization Methods and Software*, 11–12:625–653, 1999.

[TLTI08] Onur Tekdas, Jong Hyun Lim, Andreas Terzis, and Volkan Isler. Using mobile robots to harvest data from sensor fields. *to appear in IEEE Wireless Communications*, 2008.

[TMF$^+$07] Michael Todd, David Mascarenas, Eric Flynn, Tajana Rosing, Ben Lee, Daniele Musiani, Sanjoy Dasgupta, Samori Kpotufe, Daniel Hsu, Rajesh Gupta, Gyuhae Park, Tim Overly, Matt Nothnagel, and Chuck Farrar. A different approach to sensor networking for SHM: Remote powering and interrogation with unmanned aerial vehicles. In *Proceedings of the 6th International workshop on Structural Health Monitoring*, 2007.

[TPS$^+$05] Gilman Tolle, Joseph Polastre, Robert Szewczyk, David Culler, Neil Turner, Kevin Tu, Stephen Burgess, Todd Dawson, Phil Buonadonna, David Gay, and Wei Hong. A macroscope in the redwoods. In *SenSys'05: Proceedings of the 3rd international conference on Embedded networked sensor systems*, pages 51–63, 2005.

[TZA03] Lang Tong, Qing Zhao, and Srihari Adireddy. Sensor networks with mobile agents. In *MILCOM '03: Proceedings of the IEEE Military Communications Conference*, pages 688–693, 2003.

[Vaz03] Vijay V. Vazirani. *Approximation Algorithms.* Springer, 2003.

[VB00] Amin Vahdat and David Becker. Epidemic routing for partially-connected ad hoc networks. *Duke University Technical Report*, CS-2000-06, 2000.

[VKR$^+$05] Iuliu Vasilescu, Keith Kotay, Daniela Rus, Matthew Dunbabin, and Peter I. Corke. Data collection, storage, and retrieval with an underwater sensor network. In *SenSys '05: Proceedings of the 3rd international conference on Embedded networked sensor systems*, pages 154–165, 2005.

[WALR$^+$06] Geoffrey Werner-Allen, Konrad Lorincz, Mario Ruiz, Omar Marcillo, Jeff Johnson, Jonathan Lees, and Matt Welsh. Deploying a wireless sensor network on an active volcano. *IEEE Internet Computing*, 10(2):18–25, 2006.

[XRC$^+$04] Ning Xu, Sumit Rangwala, Krishna Kant Chintalapudi, Deepak Ganesan, Alan Broad, Ramesh Govindan, and Deborah Estrin. A wireless sensor network for structural monitoring. In *SenSys'04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 13–24, 2004.

[XWJL08] Guoliang Xing, Tian Wang, Weijia Jia, and Minming Li. Rendezvous design algorithms for wireless sensor networks with a mobile base station. In *MobiHoc '08: Proceedings of the 9th ACM international symposium on Mobile ad hoc networking and computing*, pages 231–240, 2008.

[XWXJ07] Guoliang Xing, Tian Wang, Zhihui Xie, and Weijia Jia. Rendezvous planning in mobility-assisted wireless sensor networks. In *RTSS '07: Proceedings of the 28th IEEE international Real-Time Systems Symposium*, pages 311–320, 2007.

[YDS95] Frances Yao, Alan Demers, and Scott Shenker. A scheduling model for reduced CPU energy. In *FOCS '95: Proceedings of the 36th Annual Symposium on Foundations of Computer Science*, pages 374–382, Washington, DC, USA, 1995. IEEE Computer Society.

[YF04] Ossama Younis and Sonia Fahmy. HEED: a hybrid, energy-efficient, distributed clustering approach for ad hoc sensor networks. *IEEE Transactions on Mobile Computing*, 3(4):366–379, Oct.-Dec. 2004.

[YOS07] Bo Yuan, Maria Orlowska, and Shazia Sadiq. On the optimal robot routing problem in wireless sensor networks. *IEEE Transactions on Knowledge and Data Engineering*, 19(9):1252–1261, 2007.

[YQ05] Lin Yuan and Gang Qu. Analysis of energy reduction on dynamic voltage scaling-enabled systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 24(12):1827–1837, 2005.

[ZA03] Wenrui Zhao and Mostafa Ammar. Message ferrying: Proactive routing in highly-partitioned wireless ad hoc networks. In *Proceedings of the IEEE Workshop on Future Trends in Distributed Computing Systems*, pages 308–314, 2003.

[ZAZ04] Wenrui Zhao, Mostafa Ammar, and Ellen Zegura. A message ferrying approach for data delivery in sparse mobile ad hoc networks. In *MobiHoc '04: Proceedings of the 5th ACM international symposium on Mobile ad hoc networking and computing*, pages 187–198, 2004.

[ZAZ05] Wenrui Zhao, Mostafa Ammar, and Ellen Zegura. Controlling the mobility of multiple data transport ferries in a delay-tolerant network. In *INFOCOM '05: Proceedings of the 24th Annual Joint Conference of the IEEE Computer and Communications Societies*, pages 1407–1418, 2005.

[ZBSF04] Bo Zhai, David Blaauw, Dennis Sylvester, and Krisztian Flautner. Theoretical and practical limits of dynamic voltage scaling. In *DAC '04: Proceedings of the 41st annual conference on Design automation*, pages 868–873, 2004.