UCSD CSE Technical Report CS2007-0911:

Data Mule Scheduling in Sensor Networks: Scheduling under Location and Time Constraints

Ryo Sugihara (ryo@ucsd.edu)
Rajesh K. Gupta (rgupta@ucsd.edu)

Computer Science and Engineering,
University of California, San Diego

October 29, 2007

Contents

Chapter 1

Introduction

Sensor networks are rapidly growing for their large applicability to various purposes. From engineering perspectives, one of the most critical issues in sensor networks is energy, because sensor networks are often deployed in remote areas where line powers are hard to obtain. It forces the sensor nodes to rely on batteries, but replacing the batteries is also hard in many cases. Moreover, sensor network applications often require long-term measurements over months. For these reasons, improving the energy efficiency is mandatory for sensor networks. Particularly, since wireless communication is one of the most energy-consuming operations on a sensor node, people have been trying to reduce communication, for example by summarizing the information inside the network.

A primary mode of communication in sensor networks is multi-hop communication, due to limited wireless communication ranges. To collect sensor data at the base station, sensor nodes forward their data to other nodes that are closer to the base station. However, multi-hop communication can be inefficient depending on how densely the sensor nodes are deployed. When the node deployment is very sparse, each hop distance becomes large and thus the large amount of energy is necessary for sending data over that distance. On the other hand, when the node deployment is very dense, the nodes close to the base station need to forward the data from many remote nodes and thus tend to run out of energy soon.

An alternative and relatively new approach for efficient data collection is to exploit the mobility. Particularly, we consider collecting data from static sensor nodes using a "data mule" via wireless communication. A data mule is a mobile node that has wireless communication capability and also a sufficient amount of storage to store the data from the sensors in the field. 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]. A data mule travels across the sensing field and collects data from each sensor node when the distance is short, and later deposits all the data to the base station. In this way, each sensor node can conserve a significant amount of energy, since it only needs to send the data over a shorter distance and has no need to forward other sensors' data all the way to the base station. In addition, as data mules return to the base station after the travel, energy issue is usually not critical for data mules.

In this paper, we are interested in the following problem: "how to control a data mule such that it collects data from all the nodes in the minimal amount of time". We call it the *data mule scheduling problem*, as we formulate it as a scheduling problem, viewing communication from each node as a job. We can control the movement of the data mule (path, speed) as well as its communication (i.e., which node it collects data from at certain time duration), where the latter corresponds to job allocation in classical scheduling problems. Despite the similarities, one of most notable differences from real-time scheduling is that the data mule scheduling problem has location constraints as well as time constraints. Availability of each job is determined by the range of wireless communication, which primarily depends on the distance from a node and thus serves as a location constraint. On the other hand, by assuming the bandwidth of wireless communication is constant, we also have a time constraint for each node necessary for transmitting the data to a data mule. The movement of data mule determines how the location constraints map to time constraints and produces different real-time scheduling problems.

Our contributions in the paper are

- Defining the data collection in sensor networks using a data mule as a scheduling problem having both location and time constraints

- Rigorous theoretical analysis on the computational complexity of the problem
- Formulating the problem as mathematical optimization problems using LP (linear programming), QP (quadratic programming), and SDP (semidefinite programming) relaxation
- Proposing a heuristic algorithm and showing its effectiveness by numerical experiments

This paper is structured as follows. In Chapter 2 we define the data mule scheduling problem and introduce related work. Chapter 3 discusses related real-time scheduling problems. In Chapter 4 we analyze two simple cases of the data mule scheduling problem and point out their similarities with speed scaling problems such as DVS (Dynamic Voltage Scaling). In Chapter 5 we discuss a more general case of the problem and prove its NP-completeness. In Chapter 6 we formulate the general problem as a nonconvex quadratic programming problem and find lower bounds of the optimal solution in two different ways including SDP relaxation. We present an efficient heuristic algorithm for the general problem in Chapter 7. Chapter 8 shows some results from numerical experiments and Chapter 9 concludes the paper.

Chapter 2

Data Mule Scheduling Problem

In this chapter we introduce the data mule scheduling problem. As we have already defined, a data mule is a mobile node that moves inside the field and collects data from each sensor. The problem is how to control the data mule so that it can collect data from the sensors in an optimal way, which we define later in the chapter.

As shown in Figure 2.1, we can decompose the problem into the following three subproblems:

1. Path selection: which trajectory the data mule follows
2. Speed control: how the data mule changes the speed during the travel
3. Job scheduling: from which sensor the data mule collects data at each time point

Path selection 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 mobility capability of data mule, there can be some constraints on the path, such as the minimum turning radius.

Speed control is the second subproblem. Once we choose the path, 2-D/3-D data mule scheduling problems are reduced to 1-D data mule scheduling problem, in which the communication ranges of each node are given as intervals on the location axis. Speed control 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 node's communication range long enough to collect all the data from it. The objective in this problem is to find an optimal (we discuss the optimality criteria later) time-speed profile while satisfying that constraint.

Final subproblem is job scheduling. Once the time-speed profile is determined, we get a mapping from each location to time point. Thus we get a 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 is to determine the allocation of time slots to jobs so that all jobs can be completed.

In this paper, we focus on the subproblems of speed control and job scheduling and leave path selection problem to our future work. The primary reason is that these two subproblems constitutes 1-D data mule scheduling problem and that is important in many cases.

2.1 Preliminaries

2.1.1 Terminology and definitions

First we define some basic terms in real-time scheduling.

- A job τ has an execution time $e(\tau)$ and a set $\mathcal{I}(\tau)$ of feasible intervals, containing one or more feasible intervals.
  - A simple job is a job with one feasible interval. A general job can have multiple feasible intervals..
- A feasible interval $I \in \mathcal{I}(\tau)$ is a time interval $[r(I), d(I)]$, where $r(I)$ is a release time and $d(I)$ is a deadline.

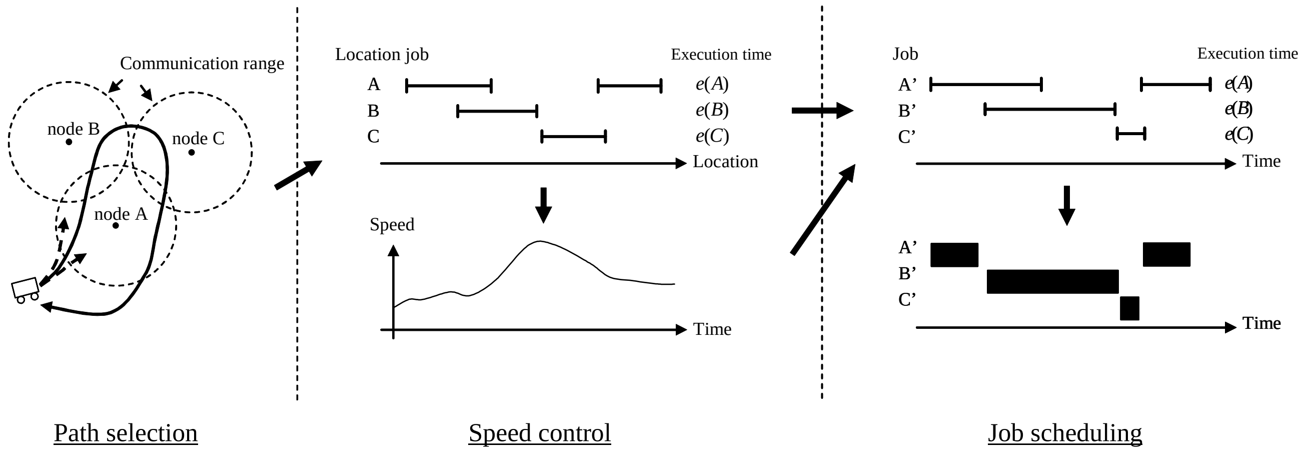


Figure 2.1: Subproblems of data mule scheduling

- A job can be executed only within its feasible intervals.

Then we define the counterpart terms in location-aware scheduling:

- A location job τ_L has an execution time $e(\tau_L)$ and a set $\mathcal{I}(\tau_L)$ of feasible location intervals, containing one or more feasible location intervals.
  - A simple location job is a location job with one feasible location interval. A general location job can have multiple feasible location intervals.
- A feasible location interval $I_L \in \mathcal{I}(\tau_L)$ is a location interval $[r(I_L), d(I_L)]$, where $r(I_L)$ is a release location and $d(I_L)$ is a deadline location.
  - A location job can be executed only within its feasible location intervals.

We may omit "location" unless it is ambiguous, and we may add "time" to the terms in real-time scheduling for clarity (e.g. "feasible time interval").

For an interval $I = [r, d]$ (also for a location interval), we define

- Length: $|I| \equiv d - r$. It is often called "relative deadline" for a simple job.
- Membership: $x \in I$ if and only if $r \leq x \leq d$.
- Containment: $I \subseteq I'$ if and only if $r' \leq r$ and $d \leq d'$ where $I' = [r', d']$.
- Intersection: $I \cap I = \emptyset$ if and only if there does not exist a point x such that $x \in I$ and $x \in I'$.

2.1.2 Assumptions

- All parameters are deterministic
- All location jobs are preemptible
- Communication bandwidth is constant within the communication range and zero out of the range.
- Each sensor has different amount of data to be collected, i.e., execution time of each location job differs.

2.2 Problem statement

We present the structure of the data mule scheduling problem in its basic form. In the following chapters we will consider several variations of the problem and give more precise formal definitions.

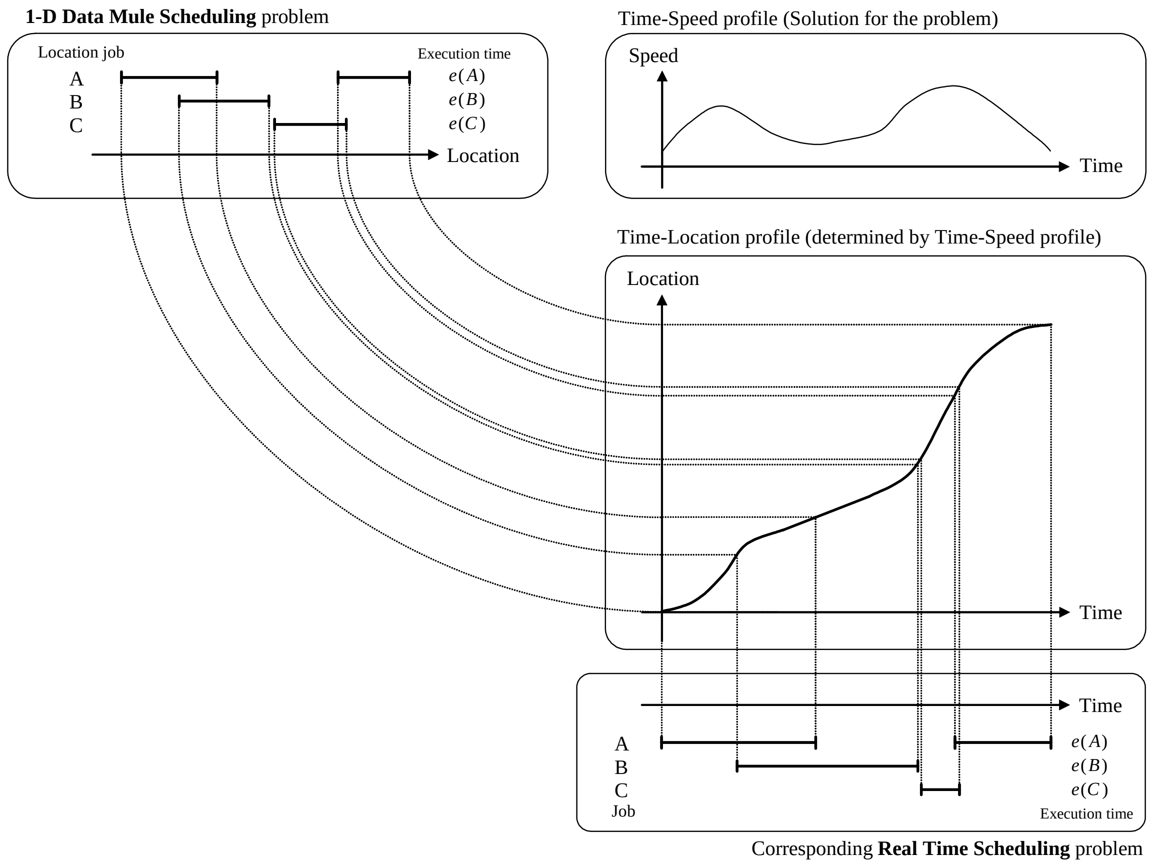

1-D Data Mule Scheduling problem
Location job
Execution time
A
B
C
e(A)
e(B)
e(C)
Location
Time-Speed profile (Solution for the problem)
Speed
Time
Time-Location profile (determined by Time-Speed profile)
Location
Time
Time
A
B
C
Job
e(A)
e(B)
e(C)
Execution time
Corresponding Real Time Scheduling problem

Figure 2.2: Main idea of data mule scheduling problem: A Time-Speed profile determines a Time-Location profile, which maps the original data mule scheduling problem to a real-time scheduling problem.

2.2.1 Instance

Input of the problem is as follows:

- A set $\mathcal{J}_L$ of location jobs, each location job $\tau_L \in \mathcal{J}_L$ has
  - Execution time $e(\tau_L)$
  - A set $\mathcal{I}(\tau_L)$ of feasible location intervals, each feasible location interval $I_L \in \mathcal{I}(\tau_L)$ has
    - Release location $r(I_L)$
    - Deadline location $d(I_L)$
- Total travel interval $[X_s, X_d]$

Each job corresponds to collecting data from each sensor node. Execution time represents the time duration required to send the data from the sensor to the data mule.

2.2.2 Objective

The objective is to find a time-speed profile and a job allocation so that the total travel time is minimized.

Another possible criteria for optimality is to maximize the amount of collected data while the total travel time is fixed. It fits a best-effort scenario that each sensor has a large amount of data and it is not possible to collect all of them within the given travel time. While it may sound reasonable, we do not choose this criteria because it has some problems in real-world scenarios. One of the main problems is that total amount of collected data does not measure the quality appropriately. More specifically, there are many application scenarios in which data collected from n different sensors are not as valuable as the same amount of data collected from $2n$ sensors. On the other hand, when an application requires data to have a temporal resolution higher than some threshold, collecting fewer amount of data from more sensors may not be a good strategy. The quality of data heavily depends on each specific application scenario and cannot be measured solely by the total amount.

It has been suggested and experimentally shown in [KSJ+04] that the speed of data mule does not affect the data rate. That is correct under the assumption that each sensor node generates data at constant rate and that the data mule periodically travels across the field. However, we argue that minimizing the total travel time is essential in at least following two cases. One case is when the periodic assumption is not valid. For example in an application scenario where sensors measure event-related data, total travel time directly affects the delay of delivery. The other case is when each sensor has limited amount of buffer, even we assume periodic travels. In that case, we need to collect data within certain period to avoid the buffers to overflow, and finding the minimum travel time helps to assess the feasibility. The same argument holds when we try to configure the data rate of each sensor so that all the data can be still collected by a data mule.

2.2.3 Constraints

The constraints for each subproblem are as follows. There will be more constraints imposed when we discuss variations of the problem in the later chapters.

- For speed control:
  - Data mule moves from the start to the destination
  - One-way movement: data mule is not allowed to move backward
- For job allocation:
  - Feasible interval: every job can be executed only within its feasible interval
  - Job completion: every job is allocated time equal to its execution time
  - Processor demand: data mule can collect data from only one node (i.e., execute only one job) at a time

2.3 Related work

The term "data mule" was coined by Shah et al. in their paper in 2003 [SRJB03]. They proposed a three-tier architecture having mobile nodes called Data MULEs (Mobile Ubiquitous LAN Extensions) in the middle tier, between wired access points and stationary sensors. As we also assumed, Data MULEs collect data from sensors when they are in close proximity and deposit it at wired access points. The difference is that they assumed Data MULEs are not controllable and move randomly, and consequently their routing scheme is rather optimistic.

The use of controlled mobility in sensor networks has been studied in several papers. Kansal et al. [KSJ+04] studied the case in which a data mule (which is called "mobile router" in the paper) periodically travels across the sensor field along a fixed path. In their model, they can only change the speed of data mule, just like our focus in this paper. They present an adaptive algorithm that changes the speed such that the amount of collected data is maximized under the constraint of maximum latency (i.e. travel time for one period). The main idea of the algorithm is to move slower when the quality of data collection is poorer and vice versa. The data mule internally classifies the nodes into two classes ("good" and "bad") depending on how much data has been successfully received from each node in the earlier periods. They also design a communication protocol based on directed diffusion [IGE00], in which the data mule issues interest messages to nodes. In this way the data mule can collect data from the nodes that are not in the direct communication range. They evaluate their algorithm on a prototype system consists of a mobile robot and motes. In some simple topologies, they show the adaptive algorithm collects more data than the one uses constant speed. Our work gives a theoretical background to their work about the computational complexity of the problem and the optimality of schedules.

Some papers discuss the path selection problem with more simplified assumptions compared to ours. For example, Somasundara et al. [SRS04] assume each sensor produces data periodically but in a different rate and the data mule needs to go to the node's exact location to collect data (i.e. no remote communication, no multihop communication). The objective is to find the path of data mule such that it can collect all the data before the buffer of each sensor overflows. They show the problem is NP-hard by a reduction from Hamiltonian cycle problem, and formulate it as an ILP (Integer Linear Programming) problem. They present some heuristic algorithms based on EDF and evaluate them by simulation experiments in terms of how many nodes miss the deadlines (i.e. buffer overflows). Ma and Yang [MY06] discuss the path selection problem in different assumptions. Their objective is 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). Different from Somasundara et al.'s paper, they consider remote wireless communication and also multihop communication among nodes. When the path of data mule is given, they show the problem of maximizing the network lifetime is formulated as a flow maximization problem that has a polynomial time algorithm. Choosing the path of data mule is done by their heuristic algorithm that uses the divide and conquer approach and finds a near-optimal path for each part of the nodes. A notable difference from our assumption is that they assume the time needed to transmit the data from each node to the data mule is negligible. They evaluate the algorithm by simulation.

Jea et al. [JSS05] studied more specialized scenario in which multiple data mules are simultaneously on the field. The paths of these data mules are fixed. Their main interest is in how to assign each sensor node to each data mule and not in how to control the motion of the data mules. The objective is to balance the load among the data mules. They assume each data mule can communicate with others by using powerful radios, and propose a distributed algorithm for load balancing. They evaluate their algorithm by a simulation on TOSSIM.

In between random mobility and controlled mobility, we can consider predictable mobility, according to the classification in [KSJ+04]. This applies to the case in which public transportation vehicles such as bus and trains are used as data mules. Chakrabarti et al. [CSA03] analyzed the gain in power consumption of exploiting predictable mobility for data collection. They modeled the data collection process as a queueing system and showed a significant reduction in power consumption.

Other than these papers on mobility in sensor networks, there are a number of papers about mobility in the context of mobile ad-hoc networks (MANETs). There are a lot of overlap between these two sets of work, but we can roughly say that sensor networks consider many-to-one data collection and MANETs consider many-to-many (i.e., peer-to-peer) communication as their objectives. Mobility in MANETs has been overcame and exploited to improve data delivery rate between each node. In addition, each node is often assumed to be mobile in MANETs, while it is usually static in sensor networks. Furthermore, a need for energy-efficiency tends to be less stringent in MANETs compared to that in sensor networks.

The classification of mobility into random, predictable, and controlled also works for MANETs. Epidemic Routing [VB00] is a routing protocol for MANETs consisting of the nodes with random mobility. It guarantees eventual message delivery by randomly exchange messages among mobile nodes. ZebraNet [JOW+02] builds upon a similar idea to improve the rate of successfully collected data in a habitat monitoring application. Though ZebraNet is usually considered a sensor network application, high mobility of nodes is closer to MANETs applications. Message Ferrying [ZAZ04] assumes controllable mobile nodes that mediate communications between sparsely deployed nodes. They also analyze this model for multiple ferries case [ZAZ05] and randomly moving nodes case [TAZ06].

Chapter 3

Related Real-time Scheduling Problem

Data mule scheduling problem is reduced to a real-time scheduling problem once we specify the time-speed profile of the data mule. More specifically, if a schedule for the corresponding real-time scheduling problem is valid, we can convert it to a valid schedule for the original data mule scheduling problem. In this chapter we present some related issues in real-time scheduling problem, specifically about feasibility testing algorithm.

3.1 Problem definition

The problem is formally defined as follows:

PREEMPTIVE SCHEDULING FOR GENERAL JOBS
INSTANCE: Set $\mathcal{J}$ of general jobs, for each general job $\tau \in \mathcal{J}$, an execution time $e(\tau) \in \mathbf{Q}^+$ and a set $\mathcal{I}(\tau)$ of feasible intervals, for each feasible interval $I \in \mathcal{I}(\tau)$, a release $r(I) \in \mathbf{Q_0^+}$ and a deadline $d(I) \in \mathbf{Q_0^+}$.
QUESTION: Is there an one-processor preemptive schedule for $\mathcal{J}$ that satisfies the release time constraints and meets all the deadlines, i.e., a one-to-one function $\sigma : \mathcal{T_S} \to \mathbf{Q_0^+}$ where $\mathcal{J_S}$ is a set of subjobs when each general job $\tau \in \mathcal{J}$ is subdivided into any number of subjobs $\tau_1, ..., \tau_k$ such that $\sum_{i=1}^{k} e(\tau_i) = e(\tau)$, $\sigma(\tau_{i+1}) \geq \sigma(\tau_i) + e(\tau_i)$, and there exists $I \in \mathcal{I}(\tau)$ such that $\sigma(\tau_i) \geq r(I)$ and $\sigma(\tau_i) + e(\tau_i) \leq d(I)$ for $1 \leq i \leq k$?

3.2 Related work

Preemptive scheduling for single processor has a long history of work. Liu and Layland [LL73] showed that Earliest Deadline First (EDF) is optimal dynamic scheduling algorithm in the sense that EDF finds a feasible schedule if and only if there is a feasible schedule.

When each job has multiple feasible intervals, however, there are only few studies. 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. Shih and colleagues [CWSK05] also 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.

3.3 Offline scheduling algorithm

3.3.1 Processor-demand-based feasibility testing

Baruah et al. [BHR93] showed the following theorem for feasibility testing based on processor demand. They considered one-processor preemptive scheduling problem for periodic tasks. Further, the relative deadline of each task can be smaller than its period (i.e., arbitrary relative deadline). Under these assumptions, the following theorem holds:

Theorem 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 $g(t_1, t_2) > t_2 - t_1$*

Processor demand $g(t_1, t_2)$ is the sum of the length of the tasks whose feasible interval is completely contained in the interval $[t_1, t_2]$. It gives the total amount of execution time required in the interval

$$g(t_1, t_2) \quad = \sum_{\substack{\tau_i \in T \\ [r_i, d_i] \in [t_1, t_2]}} e_i$$

Lemma 3.2. *For any $t_1 < t_2$ satisfying $g(t_1, t_2) > 0$, there exist $t'_1 \in \{r_i\}$ and $t'_2 \in \{d_i\}$ such that $t_1 \le t'_1 < t'_2 \le t_2$, $g(t_1, t_2) = g(t'_1, t'_2)$.*

Proof. Choose t'_1, t'_2 as follows:

$$\begin{aligned} t'_1 &= \min_{\tau_i \in T, r_i \ge t_1} \{r_i\} \\ t'_2 &= \max_{\tau_i \in T, d_i \le t_2} \{d_i\} \end{aligned}$$

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 $t_1 \le t'_1 < t'_2 \le t_2$. □

Lemma 3.3. *For any $t_1 < t_2$ and $t'_1 < t'_2$ where $t'_1 \in \{r_i\}, t'_2 \in \{d_i\}$, $g(t_1, t_2) \le t_2 - t_1$ if and only if $g(t'_1, t'_2) \le t'_2 - t'_1$.*

Proof. ("if" part) 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 3.2, 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 $t'_1 = \min_{\tau_i \in T, r_i \ge t_1}\{r_i\}$ and $t'_2 = \max_{\tau_i \in T, d_i \le t_1}\{d_i\}$. 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, $\exists t'_1 \in \{r_i\}, \exists t'_2 \in \{d_i\}, t'_1 \le t'_2 \wedge g(t'_1, t'_2) > t'_2 - t'_1$.

("only if" part) Obvious from Lemma 3.2. □

The following theorem follows immediately from Theorem 3.1 and Lemma 3.3.

Theorem 3.4. *Taskset is feasible 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 $t'_1 < t'_2$.*

3.3.2 Linear programming formulation

Using Theorem 3.4, we formulate the scheduling problem by linear programming. Since there is a polynomial time algorithm to solve linear programming, it is a polynomial time offline scheduling algorithm.

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 interval into k intervals $[s_0(= 0), s_1]$, $[s_1, s_2]$, ..., $[s_{k-1}, s_k(= T)]$, where $s_i \in P_r \cup P_d,\ s_i \le s_{i+1}$ and P_r, P_d are the set of release time and deadline, respectively. For every job $\tau \in \mathcal{J}$, consider variables $p_1(\tau), ..., p_k(\tau)$, in which $p_i(\tau)$ represents the time allocated to job τ during the interval $[s_{i-1}, s_i]$. Suppose there are n jobs and the total number of feasible intervals is w, the number of variables are $O(nw)$.

We construct a linear programming problem to find $p_i(\tau)$ for all jobs and all intervals satisfying following constraints:

- (Feasible intervals) For all $\tau \in \mathcal{J}$, $p_i(\tau) = 0$ if $[s_{i-1}, s_i] \notin \mathcal{I}(\tau)$.
- (Job completion) For all $\tau \in \mathcal{J}$, $\sum_{i=1}^{k} p_i(\tau) = e(\tau)$.
- (Processor demand) For all $1 \le i \le k$, $\sum_{\tau} p_i(\tau) \le s_i - s_{i-1}$.

Note s_i's are given constants and thus this is a linear programming problem. This construction is done in polynomial time and linear programming problem is solvable in polynomial time. After obtaining a feasible solution, we divide each interval $[s_{i-1}, s_i]$ 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.

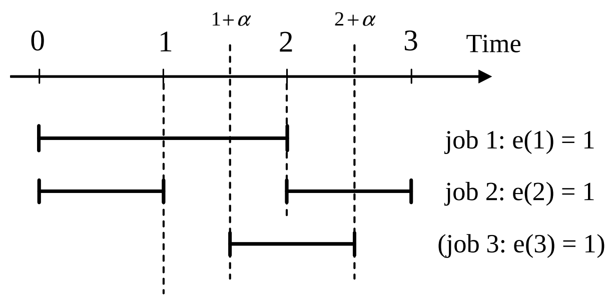


Figure 3.1: Counterexample for showing non-existence of optimal online algorithm for PREEMPTIVE SCHEDULING FOR GENERAL JOBS.

3.4 Non-existence of optimal online scheduling algorithm

In offline scheduling, information of all jobs is given to the scheduler in advance before the processor starts executing jobs. To be more practical, it would be preferable if there is an online scheduling algorithm that is optimal, such as EDF algorithm for preemptive scheduling. To be clear, 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, gets available to the scheduler when the job is first released. Unfortunately, the following theorem gives a negative result.

Theorem 3.5. *There is no optimal online scheduling algorithm for PREEMPTIVE SCHEDULING FOR GENERAL JOBS.*

Proof. Assume there exists such optimal online scheduling algorithm. We give an example set of jobs and show that, for any job allocation until a certain time, an adversary can construct a job released later that leads to a scheduling failure, yet the set of jobs as a whole remains feasible.

Figure 3.1 is a counterexample consisting of three jobs. Assume the optimal online algorithm allocates time p and q ($0 \leq p, q \leq 1, p+q \leq 1$) to job 1 and 2, respectively, within 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 3, which is released later. Assume job 3 has a feasible interval $[1+\alpha, 2+\alpha]$ and the execution time is one. Since the length of the interval is one, the whole interval needs to be allocated to job 3 to finish it. For $0 \leq p < 1$, we choose $\alpha = 1-p-\varepsilon$, where ε is a small constant satisfying $\varepsilon > 0, p+\varepsilon < 1$. Then, job 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 1 is $1-\varepsilon$, which is smaller than its execution time.

For $p = 1$, we choose $\alpha = \varepsilon$ and then job 2 cannot be finished for a similar reason. Clearly, the set of jobs remains feasible and schedulable by an offline algorithm (we can finish all three jobs by allocating $[0, \alpha]$ to job 2, $[\alpha, 1+\alpha]$ to job 1 to finish, $[1+\alpha, 2+\alpha]$ to job 3 to finish, and $[2+\alpha, 3]$ to job 2). This is a contradiction to the assumption that the algorithm is optimal. □

Chapter 4

Data Mule Scheduling: Simple Cases

We start with two simple cases for location-aware scheduling. One is constant speed, where the data mule moves at a constant speed from the start to the destination. The other is variable speed, where the data mule can freely change the speed. We present optimal algorithms for each of the variations and see some interesting similarities with speed scaling schemes such as dynamic voltage scaling (DVS).

4.1 Constant speed

4.1.1 Problem definition

The problem is formally defined as a decision problem as follows:

> **CONSTANT SPEED 1-D DATA MULE SCHEDULING**
> INSTANCE: Set $\mathcal{J}_L$ of location jobs, for each location job $\tau_L \in \mathcal{J}_L$, an execution time $e(\tau_L) \in \mathbf{Q}^+$, and a set $\mathcal{I}(\tau_L)$ of "feasible location intervals", for each feasible location interval $I_L \in \mathcal{I}(\tau_L)$, a release location $r(I_L)$ and deadline location $d(I_L)$, a start $X_s \in \mathbf{Q}_\mathbf{0}^+$, a destination $X_d \in \mathbf{Q}^+$, and a total travel time $T \in \mathbf{Q}^+$.
> QUESTION: Is there a valid schedule for the corresponding PREEMPTIVE SCHEDULING FOR GENERAL JOBS problem, if the data mule move at the constant speed $v_0 = \frac{X_d - X_s}{T}$? (i.e., Is a set $\mathcal{J}$ of jobs schedulable, where, for each job $\tau \in \mathcal{J}$, an execution time $e(\tau) = e(\tau_L)$ and a set A_τ of feasible intervals, for each feasible interval $a \in A_\tau$, a release time $r(a) = r(I_L)/v_0$ and a deadline $d(a) = d(I_L)/v_0$, where $I_L \in \mathcal{I}(\tau_L)$ and τ_L is the corresponding location job in the original problem?)

In the optimization version of the problem, the objective is to minimize T, i.e., to maximize v_0.

4.1.2 Polynomial time offline algorithms

When we assume the data mule cannot change the speed once it starts to move, there clearly does not exist an optimal online scheduling algorithm. For example, suppose 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 $(v_0 - \varepsilon)e$ (where $\varepsilon > 0$), which contradicts the assumption.

Accordingly, we provide two optimal offline scheduling algorithms, each for simple location jobs and general location jobs, respectively.

Simple location jobs

When each location job has one feasible location interval, following simple algorithm (Find-Min-MaxSpeed) finds the maximum possible v_0 such that all location jobs can be completed. Find-Min-MaxSpeed applies processor-demand based feasibility test (Theorem 3.4) for all possible pairs of a release location of one location job and a deadline location of another location job.

```
FIND-MIN-MAXSPEED($\mathcal{J}_L$)
1 for each location interval $I_L = [r(\tau'_L), d(\tau''_L)]$ s.t. $\tau'_L, \tau''_L \in \mathcal{J}_L$, $r(\tau'_L) \leq d(\tau''_L)$
2     do  ▷ Calculate processor demand for $I_L$
3         $d = \sum_{\tau_L \in \mathcal{J}_L, I(\tau_L) \subseteq I_L} e(\tau_L)$
4         $u[I_L] \leftarrow \frac{|I_L|}{d}$   ▷ Maximum possible speed allowed for $I_L$
5 return $\min_{I_L} u[I_L]$
```

where $I(\tau_L)$ is the feasible location interval for job τ_L. Note we now consider the case in which each τ_L has only one feasible interval. This algorithm runs in $O(n^3)$ time where n is the number of location jobs, but we can improve the running time 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 3.4 guarantees the feasibility iff, for all possible pairs of release time and deadline, the processor demand for the interval is equal 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, and thus the corresponding set of location jobs is feasible. 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 location jobs remains feasible.

General location jobs

For general location jobs case that 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 programming. The formulation is quite similar to the one in Section 3.3.2, except that now we map the location to time. We solve the optimization version of the problem by regarding v_0 as a variable to maximize.

We split the location interval $[X_s, X_d]$ into $(2m+1)$ location intervals $[l_0(= X_s), l_1]$, $[l_1, l_2]$, ..., $[l_{2m}, l_{2m+1}(= X_d)]$ $(l_i \leq l_{i+1})$, 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 of a feasible location interval. Then we map each location interval to a time interval using $s_i = l_i/v_0$, and obtain $(2m+1)$ non-overlapping time intervals $[s_0(= 0), s_1]$, $[s_1, s_2]$, ..., $[s_{2m}, s_{2m+1}(= T)]$. Note each s_i is a variable, since v_0 is a variable. In the same way, we map each location job in $\mathcal{J}_L$ to a job by mapping each feasible location interval of the job to a feasible time interval, and obtain a new set of jobs $\mathcal{J}$.

For every 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 during the time interval $[s_i, s_{i+1}]$. Equivalently, $p_i(\tau)$ represents the time allocated to job τ_L within the location interval $[l_i, l_{i+1}]$.

We construct a linear programming problem as follows:

Constant speed, general location jobs

Variables

- v_0: speed of data mule
- $p_i(\tau)$ $(0 \leq i \leq 2m)$: time allocated to job τ in interval $[s_i, s_{i+1}]$ (or equivalently, time allocated to location job τ_L in location interval $[l_i, l_{i+1}]$)

Objective Maximize v_0

Constraints

- (Positiveness) $p_i(\tau) \geq 0$
- (Feasible intervals) For all $\tau \in \mathcal{J}$, if $[l_i, l_{i+1}] \notin \mathcal{I}(\tau_L)$,

$$p_i(\tau) = 0 \tag{4.1}$$

  where $\tau_L \in \mathcal{J}_L$ is mapped to $\tau \in \mathcal{J}$.

- (Job completion) For all $\tau \in \mathcal{J}$,

$$\sum_{i=0}^{2m} p_i(\tau) = e(\tau) \quad (= e(\tau_L)) \tag{4.2}$$

- (Processor demand) For all $0 \leq i \leq 2m$,

$$\begin{aligned} \sum_{\tau \in \mathcal{J}} p_i(\tau) &\leq s_{i+1} - s_i \\ &= \frac{l_{i+1} - l_i}{v_0} \end{aligned} \tag{4.3}$$

The constraint on processor demand is converted to a linear constraint by introducing a new variable $u_0 = \frac{1}{v_0}$ instead of v_0. This algorithm is correct and optimal from a similar argument as single feasible interval case above.

4.2 Variable speed

4.2.1 Problem definition

Variable speed case of the data mule scheduling problem is formally defined as follows:

> **VARIABLE SPEED 1-D DATA MULE SCHEDULING**
> INSTANCE: Set $\mathcal{J}_L$ of location jobs, for each location job $\tau_L \in \mathcal{J}_L$, an execution time $e(\tau_L) \in \mathbf{Q}^+$, and a set $\mathcal{I}(\tau_L)$ of "feasible location intervals", for each feasible location interval $I_L \in \mathcal{I}(\tau_L)$, a release location $r(I_L)$ and deadline location $d(I_L)$, a start $X_s \in \mathbf{Q}_\mathbf{0}^+$, a destination $X_d \in \mathbf{Q}^+$, a speed range $[v_{min}, v_{max}]$, and a total travel time $T \in \mathbf{Q}^+$.
> QUESTION: Is there set S of speed changing points, consisting of $2m$ points where m is the total number of feasible location intervals for all location jobs, that characterizes the travel from X_s to X_d satisfying the speed constraint, such that there exists a satisfying schedule for the corresponding PREEMPTIVE SCHEDULING FOR GENERAL JOBS problem? (i.e., Each speed changing point $s \in S$ is a tuple of a location $x(s) \in \mathbf{Q}_\mathbf{0}^+$, a time $t(s) \in \mathbf{Q}_\mathbf{0}^+$, and a speed $v(s) \in \mathbf{Q}_\mathbf{0}^+$ satisfying $v_{min} \leq v(s) \leq v_{max}$. S is sorted by $t(s)$ and satisfying $x(s_{i+1}) - x(s_i) = v(s_i)(t(s_{i+1}) - t(s_i))$ for $1 \leq i < 2m$. Define time-speed profile $v(t)$ by a piecewise constant function $v(t) = v(s_k)$ where k is an integer satisfying $t(s_k) \leq t < t(s_{k+1})$. Define time-location profile $x(t)$ by $x(t) = \int_0^t v(t)dt$. For functions $f : P_r \to \mathbf{Q}_\mathbf{0}^+$, $g : P_d \to \mathbf{Q}_\mathbf{0}^+$ where $P_r = \bigcup_{\tau_L \in \mathcal{J}_L} \bigcup_{I_L \in \mathcal{I}(\tau_L)} r(I_L)$, $P_d = \bigcup_{\tau_L \in \mathcal{J}_L} \bigcup_{I_L \in \mathcal{I}(\tau_L)} d(I_L)$, $f(r(I_L)) = \min S_t(r(I_L))$, $g(d(I_L)) = \max S_t(d(I_L))$, and $S_t(y) = \{t | x(t) = y\}$, is a set $\mathcal{J}$ of jobs schedulable, where, for each job $\tau \in \mathcal{J}$, an execution time $e(\tau) = e(\tau_L)$ and a set $\mathcal{I}(\tau)$ of feasible intervals, for each feasible interval $I \in \mathcal{I}(\tau)$, a release time $r(I) = f(r(I_L))$ and a deadline $d(I) = g(d(I_L))$, where $I_L \in \mathcal{I}(\tau_L)$ and τ_L is the corresponding location job in the original problem?)

Functions f and g represent mapping functions from location to a time point. We need two separate functions for release locations and deadline locations, since a single location can map to a time interval when the data mule stops at the location. In this case, release location maps to the beginning of the time interval and deadline location maps to the end of it.

We can restrict $v(t)$ to be piecewise constant, since any speed change within an interval of neighboring two points in $P_r \cup P_d$ can be replaced by a constant speed without affecting the corresponding real-time scheduling problem and thus its feasibility.

4.2.2 Polynomial time algorithms

Simple location jobs

When $v_{min} = 0$, we show the following EDF-based algorithm is an optimal online algorithm that minimizes the total travel time. In the algorithm, EDF-WITH-STOP, the data mule moves at v_{max} while executing a job having the earliest deadline, just in the same way as ordinary EDF 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.

```
EDF-WITH-STOP($\mathcal{J}_L$)
 1  ▷ Init: Set of active jobs: $\mathcal{J}_A$, init with $\emptyset$
 2  ▷        Data mule's speed: $v$, init with $v_{max}$
 3  ▷        Time allocated to jobs: $a(\tau_L)$ for all $\tau_L \in \mathcal{J}_L$, init with zero
 4  ▷        Current location: $x_c$, init with $X_s$
 5  On $\exists \tau_L \in \mathcal{J}_A,\ d(\tau_L) = x_c \wedge a(\tau_L) < e(\tau_L)$:    ▷ Unfinished jobs at deadline
 6     $\mathcal{J}_0 \leftarrow \{\tau_L | \tau_L \in \mathcal{J}_A, d(\tau_L) = x_c \wedge a(\tau_L) < e(\tau_L)\}$
 7     $v \leftarrow 0$              ▷ Stop
 8     Complete each job in $\mathcal{J}_0$
 9     $\mathcal{J}_A \leftarrow \mathcal{J}_A \setminus \mathcal{J}_0$
10     $v \leftarrow v_{max}$         ▷ Move at $v_{max}$ again
11     $\tau_{ed} \leftarrow \arg\min_{\tau_L \in \mathcal{J}_A} d(\tau_L)$    ▷ Job with earliest deadline
12     Execute $\tau_{ed}$

13  On $\exists \tau_L \in \mathcal{J}_L,\ r(\tau_L) = x_c$           ▷ Job released
14     $\mathcal{J}_A \leftarrow \mathcal{J}_A \cup \{\tau_L | r(\tau_L) = x_c\}$
15     $\tau_{ed} \leftarrow \arg\min_{\tau_L \in \mathcal{J}_A} d(\tau_L)$
16     Execute $\tau_{ed}$

17  On $\exists \tau_L \in \mathcal{J}_A,\ a(\tau_L) = e(\tau_L)$        ▷ Job completed
18     $\mathcal{J}_A \leftarrow \mathcal{J}_A \setminus \tau_L$
19     $\tau_{ed} \leftarrow \arg\min_{\tau_L \in \mathcal{J}_A} d(\tau_L)$
20     Execute $\tau_{ed}$
```

We have a following theorem about the optimality of EDF-WITH-STOP.

Theorem 4.1. EDF-WITH-STOP *is optimal for VARIABLE SPEED 1-D DATA MULE SCHEDULING for simple location jobs when* $v_{min} = 0$

Proof. Every valid schedule that uses the speed between 0 and v_{max} can be converted to the one that only uses 0 and v_{max}. Thus, for a valid schedule, the time to stop is minimized if and only if the schedule is optimal. Further, for a valid and reasonable 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 hard real-time scheduling problem in which we want to maximize the allocated time, or equivalently to minimize the idle time. Since it is a hard real-time scheduling, we do not allow the processor to execute a job after its deadline, and thus some jobs may be left unfinished. However, a non-standard assumption is that partial job execution counts in this problem. We claim the following algorithm similar to EDF is optimal for the problem:

Algorithm: At any time, allocate a job with the earliest deadline from the set of available jobs

Note this algorithm is identical to EDF when the system is underloaded. We show this algorithm minimizes the idle time by converting from an optimal schedule. Let 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 compare A and A_{opt} from the beginning and swap allocations in A_{opt} as follows when they differ:

- Case 1: $A_{opt}(t_a, t_b) = \tau_1$, $A(t_a, t_b) = \tau_2$, $\tau_1 \neq \tau_2$
  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

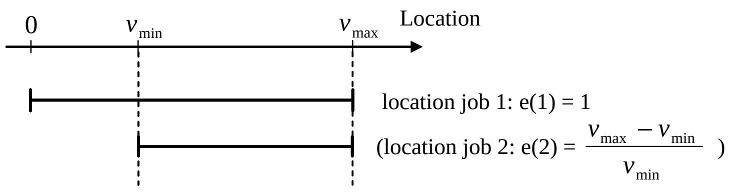

0
$v_{\min}$
$v_{\max}$
Location
location job 1: e(1) = 1
(location job 2: e(2) = $\frac{v_{\max} - v_{\min}}{v_{\min}}$)

Figure 4.1: Counterexample for showing non-existence of optimal online algorithm for VARIABLE SPEED 1-D DATA MULE SCHEDULING for simple location jobs when $v_{min} > 0$.

τ_2 by the time t_a in A_{opt} is shorter by $(t_b - t_a)$ than that in A, and thus τ_2 is not finished yet in A_{opt}. For the same reason, we can make a list of pairs $L = \{(t'_a, t'_b)\}$ such that $\sum_{(t'_a,t'_b)\in L}(t'_b - t'_a) = t_b - t_a$. 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. It is possible because the time t'_b is before the deadline of τ_1, since $t'_b \leq d(\tau_2) \leq d(\tau_1)$ (because of EDF-based allocation).

- Case 2: $A_{opt}(t_a, t_b) = \emptyset$, $A(t_a, t_b) = \tau$
  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 $A_{opt}(t'_a, t'_b) = \tau_1$ and $A_{opt}(t_a, t_b) = \tau_2$ 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: $A_{opt}(t_a, t_b) = \tau$, $A(t_a, t_b) = \emptyset$
  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 at t_a. However, it is a contradiction, since τ is available at t_a and the algorithm allocates time to a job whenever there are any available jobs.

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

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

Theorem 4.2. *There is no optimal online scheduling algorithm for VARIABLE SPEED 1-D DATA MULE SCHEDULING for simple location jobs when $v_{min} > 0$*

Proof. Assume there exists such optimal online algorithm. Figure 4.1 shows an example we use. We assume the total travel interval is $[0, v_{max}]$. Let p denote the time the optimal algorithm spent on moving from location 0 to v_{min}. Since the range of speed is $[v_{min}, v_{max}]$, $\frac{v_{min}}{v_{max}} \leq p \leq 1$. For an optimal algorithm, we can assume the whole time is spent on processing location job 1, which is the only available job.

When $p < 1$, the adversary releases location job 2, which has feasible location interval $[v_{min}, v_{max}]$ and execution time $\frac{v_{max}-v_{min}}{v_{min}}$. Since the data mule can spend at most $\frac{v_{max}-v_{min}}{v_{min}}$ seconds to move from v_{min} to v_{max}, it needs to process location job 2 for the whole time to finish it, and thus it is impossible to finish both job 1 and 2. The set of jobs is schedulable by an offline algorithm: moving at v_{min} all the time, processing job 1 until completion and then job 2.

When $p = 1$, the adversary does not release location job 2. Then the total travel time is at least $1+\frac{v_{max}-v_{min}}{v_{max}}$, which is strictly larger than 1 second. However, an optimal offline schedule can reduce the total travel time to 1 second, by moving at v_{max} all the time and finishing location job 1.

□

We have the following optimal offline algorithm, based on Yao et al.'s algorithm [YDS95] for dynamic voltage scaling. The algorithm is based on processor-demand analysis and uses FIND-MIN-MAXSPEED (in Section 4.1.2) internally.

```
repeat
    $v_c \leftarrow$ FIND-MIN-MAXSPEED($\mathcal{J}_L$)
    if $v_c < v_{min}$
        then return INFEASIBLE
    elseif $v_c > v_{max}$
        then Set $v_{max}$ for all remaining intervals and Finish
        else
            Set $v_c$ for the current critical interval
            Remove jobs within the critical interval
            Compress remaining jobs (as in [YDS95])
```

Here is how this algorithm works. FIND-MIN-MAXSPEED finds a critical interval and the corresponding speed. It is the minimum speed that makes the processor demand for all location intervals equal or less than the time allocated to that interval. In other words, if the data mule moves at the speed more than v_c, there is at least one interval in which the time is less than the processor demand (thus violates the feasibility). Therefore, if $v_c < v_{min}$ (Line 3), it is infeasible. In each iteration, v_c is nondecreasing. This is shown by the same reasoning as 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 6).

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.

General location jobs

When a job may have multiple feasible location intervals, the following theorem states there is no optimal online algorithm. It is proved in a similar way as Theorem 3.5.

Theorem 4.3. *There is no optimal online scheduling algorithm for VARIABLE SPEED 1-D DATA MULE SCHEDULING for general location jobs*

Proof. Assume there exists such optimal online scheduling algorithm. We give an example set of location jobs and show that, for any job allocation until a certain location, we can construct a location job released later that makes the total travel time take longer than that of the optimal offline algorithm.

Figure 4.2 is an example consists of three location jobs. Assume the algorithm allocates time p and q ($p, q \geq 0,\ p + q \leq 1$) to location job 1 and 2, respectively, within the location interval $[0, v_{max}]$. We assume $q = 1 - p$ since the algorithm is optimal and the speed is v_{max} all the time without any idle time in the optimal offline algorithm, as we see later. Note that, at location v_{max}, the algorithm has no information about location job 3, which is released afterwards. Assume location job 3 has a feasible location interval $[(1 + \alpha)v_{max}, (2 + \alpha)v_{max}]$ $(0 < \alpha \leq 1)$ and the execution time is one. Since the length of the feasible location interval equals to v_{max}, the whole location interval needs to be allocated to location job 3 to finish it while moving in speed v_{max}.

The optimal (i.e., minimum) travel time in this example is 3 seconds. It is achieved by moving at speed v_{max} all the time, while allocating in order as follows: α second to job 2, 1 second to job 1 to finish, 1 second to job 3 to finish, and $(1 - \alpha)$ second to job 2 to finish all the location jobs at location $3v_{max}$.

For the optimal online algorithm we assume, however, an adversary can easily choose α such that the total travel time is strictly greater than 3 seconds. Specifically, if the adversary chooses any value so that $\alpha \neq 1 - p$, the data mule either needs to decrease the speed (to finish location job 1 or 2) or moves without executing any jobs (because there is no unfinished location job available) in the location intervals $[v_{max}, (1 + \alpha)v_{max}]$ and $[(2 + \alpha)v_{max}, 3v_{max}]$. In either case, the total travel time is greater than 3 seconds, which contradicts the assumption. □

As for offline algorithms, we can achieve the optimality by using linear programming. We omit the details of the formulation here, as it is quite similar to the one for constant speed case in the previous section, except that we add the speed of data mule v_i for each location interval $[l_i, l_{i+1}]$ as variables.

When $v_{min} > 0$, we can construct a linear programming problem as follows:

Variable speed, general location jobs, $v_{min} > 0$

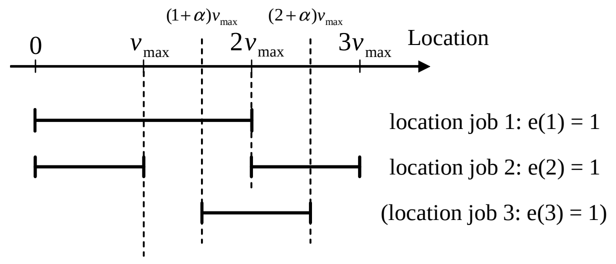


Figure 4.2: Counterexample for showing non-existence of optimal online algorithm for VARIABLE SPEED 1-D DATA MULE SCHEDULING for general location jobs.

Variables For each location interval $[l_i, l_{i+1}]$ $(0 \leq i \leq 2m)$,

- v_i: speed of data mule
- $p_i(\tau)$: time allocated to job τ

Objective Minimize the total travel time

$$\sum_{i=0}^{2m} \frac{l_{i+1} - l_i}{v_i} \tag{4.4}$$

Constraints

- (Speed)

$$v_{min} \leq v_i \leq v_{max} \tag{4.5}$$

- (Feasible intervals) For all $\tau \in \mathcal{J}$, if $[l_i, l_{i+1}] \notin \mathcal{I}(\tau_L)$

$$p_i(\tau) = 0 \tag{4.6}$$

  where $\tau_L \in \mathcal{J}_L$ is mapped to $\tau \in \mathcal{J}$.

- (Job completion) For all $\tau \in \mathcal{J}$,

$$\sum_{i=0}^{2m} p_i(\tau) = e(\tau) \tag{4.7}$$

- (Processor demand)

$$\sum_{\tau \in \mathcal{J}} p_i(\tau) \leq \frac{l_{i+1} - l_i}{v_i} \tag{4.8}$$

We can eliminate $\frac{1}{v_i}$ terms from (4.4) and (4.8) by introducing new variables $u_i = \frac{1}{v_i}$ instead of v_i. From (4.5), the range of u_i is $\frac{1}{v_{max}} \leq u_i \leq \frac{1}{v_{min}}$. Now the objective and all the constraints are linear to the variables. This algorithm is correct and optimal from a similar argument as single feasible interval case above.

When $v_{min} = 0$, there is a subtle problem in the above formulation in its original form, since it contains $\frac{1}{v_i}$ terms. We can accommodate this case by using the variables $d_i = \frac{l_{i+1} - l_i}{v_i}$ that represents the time to stay within the interval $[l_i, l_{i+1}]$[1]. Then we can change the formulation as follows:

[1]Strictly speaking, interpreting d_i as the time to stay within the interval is not correct. When $l_i = l_{i+1}$ and $d_i > 0$ (i.e., the data mule stops at l_i), the actual time to stay within the interval $[l_{i-1}, l_i]$ is $(d_{i-1} + d_i)$ if we include the boundary points. However, this is only an issue of interpretation, and the formulation is correct due to the constant speed assumption within each interval.

- the objective is minimizing

$$\sum_{i=0}^{2m} d_i \tag{4.9}$$

- the speed constraint (4.5) is rewritten as

$$\frac{l_{i+1} - l_i}{v_{max}} \leq d_i \tag{4.10}$$

  Note we do not have an upper bound.

- the processor demand constraint (4.8) is rewritten as

$$\sum_{\tau \in \mathcal{J}} p_i(\tau) \leq d_i \tag{4.11}$$

Notice that, when $v_{min} = 0$, even a job only with zero-length feasible location intervals (i.e., $r(I) = d(I)$ for all $I \in \mathcal{I}(\tau_L)$) can be scheduled by making the data mule stop at the location to execute the job. We can handle this situation as well by changing the formulation as above.

4.3 Similarity with speed scaling problem

In the data mule scheduling, we map each location to time point by determining the speed of the data mule and obtain corresponding real-time scheduling problems. Location quantities such as release and deadline locations are the variables dependent on the speed and time quantity (execution time) does not change. Conversely, we can map time points to locations: release and deadline locations are unchanged and execution time changes according to the speed of the data mule. Interestingly enough, the resulting problem is analogous to speed scaling problem such as DVS (dynamic voltage scaling), which is a popular technique for power management in embedded systems. Here we show the correspondence between data mule scheduling problems and speed scaling problems.

4.3.1 Constant speed

The constant speed case corresponds to static speed scaling (SSS) scheme, which is defined as follows:

An instance of SSS problem is $\{\mathcal{J}_S, [s_{min}, s_{max}]\}$, where

- $\mathcal{J}_S$ is a set of jobs, for each job τ in $\mathcal{J}_S$
  - $r(\tau), d(\tau)$: release time and deadline
  - $e_c(\tau)$: execution time in CPU cycle count. When the processing speed is s, execution time $e(\tau)$ is $e_c(\tau)/s$.
- s_{min}, s_{max}: lower bound and upper bound of processing speed, represented in CPU cycles per unit time. s_{max} is usually determined by hardware limitation. s_{min} is a sufficiently small value.

Assuming preemptive scheduling and that the processing speed is continuous, the objective is to find a constant processing speed s_0 ($s_{min} \leq s_0 \leq s_{max}$) that keeps the set of jobs feasible, while minimizing the energy consumption $E(s_0)$ defined as follows:

$$E(s_0) \quad = \quad \int_{t_0}^{t_1} P(s_0)dt \tag{4.12}$$

where t_0 is the time the first job released, t_1 is the time the last job completes, and P is the power, i.e., the energy consumed per unit time. When P is a convex function of the processor speed, as is generally assumed, the objective is equivalent to minimizing s_0.

Now let's consider an instance of constant speed data mule scheduling problem $\{\mathcal{J}_L, [v_{min}, v_{max}]\}$, where $\mathcal{J}_L$ is a set of location jobs. When the speed is v_0, a location job τ_L is mapped to a job τ that has a release time $r(\tau) = \frac{r(\tau_L)}{v_0}$, a deadline $d(\tau) = \frac{d(\tau_L)}{v_0}$, and an execution time $e(\tau) = e(\tau_L)$. The objective is to minimize the total travel time, which is equivalent to maximizing the speed v_0. By expanding the time axis by a factor v_0, we can convert the original problem to an equivalent problem $\{\mathcal{J}', [v'_{min}, v'_{max}]\}$, where each job τ' in $\mathcal{J}'$ has a release time $r(\tau') = r(\tau_L)$, a deadline $d(\tau') = d(\tau_L)$, and an execution time $e(\tau') = v_0 e(\tau)$. The speed constraint becomes $[\frac{1}{v_{max}}, \frac{1}{v_{min}}]$ and the objective is unchanged: to maximize v_0, or equivalently, to minimize $\frac{1}{v_0}$, while keeping $\mathcal{J}'$ feasible.

Now we can see the similarity between the static speed scaling and the constant speed data mule scheduling: s_0 in the former problem corresponds to $1/v_0$ in the latter problem, and we can map an instance of one problem to an instance of the other as follows:

- From constant speed data mule scheduling to SSS:

$$\{\{r(\tau_L), d(\tau_L), e(\tau_L)\}, [v_{min}, v_{max}]\} \quad \rightarrow \quad \left\{\{r(\tau_L), d(\tau_L), e(\tau_L)\}, \left[\frac{1}{v_{max}}, \frac{1}{v_{min}}\right]\right\} \tag{4.13}$$

- From SSS to constant speed data mule scheduling:

$$\{\{r(\tau), d(\tau), e_c(\tau)\}, [s_{min}, s_{max}]\} \quad \rightarrow \quad \left\{\{r(\tau), d(\tau), e_c(\tau)\}, \left[\frac{1}{s_{max}}, \frac{1}{s_{min}}\right]\right\} \tag{4.14}$$

4.3.2 Variable speed

A natural extension of the above argument is to compare the variable speed case with dynamic speed scaling (DSS) scheme. However, the correspondence between these two are not as clear as the previous case.

An instance of a DSS problem is $\{\mathcal{J}_S, [s_{min}, s_{max}]\}$, which is same as SSS problem. The output is $s(t)$, a function representing the change of speed. Based on the discussion in [YDS95], we can constrain $s(t)$ to be pointwise 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. A DSS problem is an optimization problem defined as follows:

Dynamic speed scaling

Variables For each interval $[T_i, T_{i+1}]$,

- s_i: processor speed in CPU cycles per unit time
- $a_i(\tau)$: CPU cycle counts allocated to job τ

where $T_i = \sum_{k=0}^{i} t_k$.

Objective Minimize the energy consumption:

$$\sum_i P(s_i) t_i \tag{4.15}$$

where $P(x)$ is a function representing the power consumption for processor speed x.

Constraints

- (Minimum/maximum speed)

$$s_{min} \leq s_i \leq s_{max} \tag{4.16}$$

- (Feasible interval) For all $\tau \in \mathcal{J}_S$, if $[T_i, T_{i+1}] \in \mathcal{I}(\tau)$

$$a_i(\tau) = 0 \tag{4.17}$$

- (Job completion) For all $\tau \in \mathcal{J}_S$

$$\sum_i a_i(\tau) \quad = \quad e_c(\tau) \tag{4.18}$$

- (Processor demand)

$$\sum_{\tau \in \mathcal{J}_S} a_i(\tau) \leq s_i t_i \tag{4.19}$$

Similarly as the constant speed case, s_i in DSS problem corresponds to $1/v_i$ in variable speed data mule scheduling problem, when we set the power function $P(s) = s$. It implies that variable speed data mule scheduling for simple location jobs is easier than general dynamic speed scaling 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, variable speed data mule scheduling problem is not as easy as this case, since $s = 0$ in speed scaling problem implies infinite speed, 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 (Optimal-Schedule) [YDS95], since it only assumes $P(s)$ to be a convex function of s. Our algorithm presented in Section 4.2.2 for simple location jobs is based on Yao et al.'s algorithm.

Chapter 5

Data Mule Scheduling: General Case

In the previous chapter, we have analyzed two simple cases of data mule scheduling problem. The first case, constant speed case, is simple but does not fully exploit the data mule's capability to change the speed dynamically. The second, variable speed case, is a better model in that sense, but is still not realistic, since it is assumed that the data mule can change the speed instantly, which implies infinite acceleration.

In this chapter we consider a more general and realistic case of the data mule scheduling problem. Specifically, we assume there is a constraint on the acceleration of data mule. This formulation is general in terms of that it contains the two simple cases. We give a formal definition of the problem and show its NP-completeness.

5.1 Variable speed with acceleration constraint

The general case of the data mule scheduling problem is formally defined as follows:

GENERALIZED 1-D DATA MULE SCHEDULING
INSTANCE: Set $\mathcal{J}_L$ of location jobs, for each location job $\tau_L \in \mathcal{J}_L$, an execution time $e(\tau_L) \in \mathbf{Q}^+$, and a set $\mathcal{I}_L$ of feasible location intervals, for each feasible location interval $I_L \in \mathcal{J}_L$, a release location $r(I_L)$ and deadline location $d(I_L)$, a maximum absolute acceleration $a_{max} \in \mathbf{Q}_\mathbf{0}^+$, a start $X_s \in \mathbf{Q}_\mathbf{0}^+$, a destination $X_d \in \mathbf{Q}^+$, and a total travel time $T \in \mathbf{Q}^+$.
QUESTION: Is there a set S of acceleration changing points, consisting of up to $6m + 4$ points where m is the total number of feasible intervals for all jobs, that characterizes the travel from X_s to X_d satisfying the one-way movement constraint and the maximum acceleration constraint such that there exists a valid schedule for the corresponding PREEMPTIVE SCHEDULING FOR GENERAL JOBS problem? (i.e., Each acceleration changing point $s \in S$ is characterized by a location $x(s) \in \mathbf{Q}_\mathbf{0}^+$, a time duration $z(s) \in \mathbf{Q}_\mathbf{0}^+$, a speed $v(s) \in \mathbf{Q}_\mathbf{0}^+$, and an acceleration $a(s) \in \mathbf{Q}$, satisfying $v(s) \geq 0$ and $-a_{max} \leq a(s) \leq a_{max}$. S is sorted by $x(s)$ and, for $1 \leq i < |S|$, $v(s_{i+1}) - v(s_i) = a(s_i)z(s_i)$ and $x(s_{i+1}) - x(s_i) = \frac{1}{2}a(s_i)z(s_i)^2 + v(s_i)z(s_i)$. Define time-speed profile $v(t)$ by a continuous function $v(t) = v(s_k) + a(s_k)(t - t_k)$ over $t \in [0, T]$ where $t_i = \sum_{j=1}^{i-1} z(s_j)$ and k is an integer satisfying $t_k \leq t \leq t_{k+1}$. Define time-location profile $x(t)$ by $x(t) = \int_0^t v(t)dt$. For functions $f : P_r \to \mathbf{Q}_\mathbf{0}^+$, $g : P_d \to \mathbf{Q}_\mathbf{0}^+$ where $P_r = \bigcup_{\tau_L \in \mathcal{J}_L} \bigcup_{I_L \in \mathcal{I}(\tau_L)} r(I_L)$, $P_d = \bigcup_{\tau_L \in \mathcal{J}_L} \bigcup_{I_L \in \mathcal{I}(\tau_L)} d(I_L)$, $f(r(I_L)) = \min S_t(r(I_L))$, $g(d(I_L)) = \max S_t(d(I_L))$, and $S_t(y) = \{t | x(t) = y\}$, a set $\mathcal{J}$ of jobs is schedulable, where, for each job $\tau \in \mathcal{J}$, an execution time $e(\tau) = e(\tau_L)$ and a set $\mathcal{I}(\tau)$ of feasible intervals, for each feasible interval $I \in \mathcal{I}(\tau)$, a release time $r(I) = f(r(I_L))$ and a deadline $d(I) = g(d(I_L))$, where $I_L \in \mathcal{I}(\tau_L)$ and τ_L is the corresponding location job in the original problem?)

where $\mathbf{Q}$ is the set of rational numbers. $\mathbf{Q}_\mathbf{0}^+$ means nonnegative rational numbers and $\mathbf{Q}^+$ means positive rational numbers. When $a_{max} = 0$, it is the constant speed case. Similarly, when $a_{max} = +\infty$, it is the variable speed case.

The constraint on the size of S comes from the following lemma:

Lemma 5.1. *For any satisfying set S' of acceleration changing points for a given instance of GENERALIZED 1-D DATA MULE SCHEDULING, there exists a satisfying set S of acceleration changing points that satisfies*

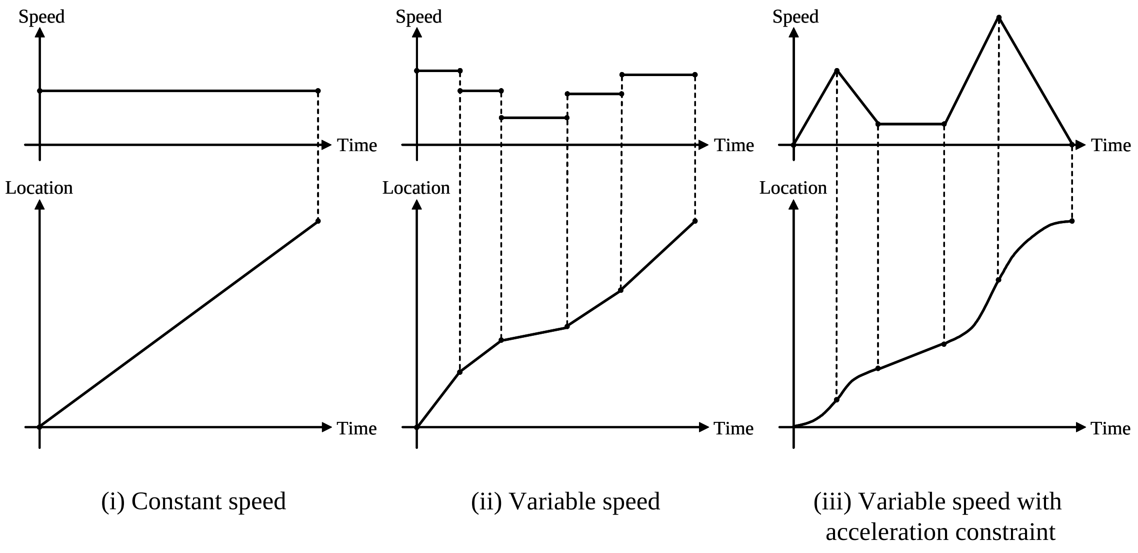


Figure 5.1: Comparison of different variations of the problem: (i) Constant speed (Section 4.1), (ii) Variable speed (Section 4.2), and (iii) Variable speed with acceleration constraint (this chapter)

$S \leq 6m + 4$*, where* m *is the total number of feasible location intervals for all jobs.*

Proof idea. Since each feasible interval has a release location and deadline location, $|P_r \cup P_d| \leq 2m$. Let's call X_s, X_d, and these locations "constrained points" in the sense that time and speed at these locations are uniquely determined by S'. There are up to $2m + 2$ constrained points. Moreover, if any two travels agree time and speed at all these constrained points, they are "equivalent": i.e., they map the original problem to the same Preemptive Scheduling with Multiple Feasible Intervals problem. By considering an equivalent movement (See Section 6.1.2 for details), any valid partial travel between two locations has an equivalent partial travel that has up to two acceleration changing points in between (edge points excluded) and acceleration is either a_{max}, $-a_{max}$, or 0. By converting a travel in this way, the resulting travel has up to $6m + 4$ acceleration changing points.

For convenience, we list some physics formula that we use afterwards. Let the initial speed v_0 at location x_0. After moving in an acceleration a for time t, the data mule has the speed v at location x. Then the following relations hold:

- Speed and time: $v = at + v_0$
- Location and time: $x = \frac{1}{2}at^2 + v_0 t + x_0$
- Location and speed: $v^2 - v_0^2 = 2a(x - x_0)$ (i.e., $v = \sqrt{v_0^2 + 2a(x - x_0)}$)

5.2 NP-completeness proofs

Obviously there is no optimal online algorithm for GENERALIZED 1-D DATA MULE SCHEDULING by considering a location job having a zero-length feasible location interval. As for offline algorithms, we show in this section that the problem is NP-complete for general location jobs[1]. We first show the membership in NP and then prove NP-hardness for general location jobs, particularly when k, the number of feasible location intervals per task, is fixed ($k \geq 2$) and arbitrary. The NP-hardness proofs are by reductions from PARTITION and 3-PARTITION, respectively. Although the constructions are different for each case, they are based on similar ideas as listed below:

[1]Complexity for simple location jobs is open as for now.

- Map each binary choice in the original problems to a "stop/skip" choice in the data mule scheduling problem.
  - Use a location job with two zero-length feasible location intervals at separate locations: the data mule needs to stop at one of these locations.
  - Note the data mule can still 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.

5.2.1 Membership in NP

Lemma 5.2. *GENERALIZED 1-D DATA MULE SCHEDULING is in NP.*

Proof. We show we have a polynomial time verifier for the problem. A solution consists of a set S of acceleration changing points and a set T of job allocation. An acceleration changing point $s \in S$ is represented as a quadruple $\{x(s), z(s), v(s), a(s)\}$, where $x(s) \in \mathbf{Q_0^+}$ is the current location, $z(s) \in \mathbf{Q_0^+}$ is the duration, $v(s) \in \mathbf{Q_0^+}$ is the current speed, and $a(s) \in \mathbf{Q}$ is the acceleration. Without loss of generality, we assume S is sorted by $x(s)$ (current location) in the ascending order. Tie breaking is arbitrary but s with nonzero $a(s)$ must be the last of tuples with same $x(s)$. An allocation $t \in T$ is represented as a triple $\{t_s(t), t_e(t), id(t)\}$, where $t_s(t), t_e(t)$ are the start and end of the duration, respectively, and $id(t)$ is the ID of the job that the duration is allocated.

We first verify that S represents a valid travel. By "valid travel", we mean that the data mule can make continuous movements by following S in order. For example, a travel is not valid if the location/speed of an acceleration changing point specified in S is different from actual location/speed after the data mule moved as instructed in the previous acceleration changing point. The following procedures verifies S is valid:

- Repeat for all $s \in S$:
  - Calculate x', v' as follows: $x' \leftarrow x(s) + \frac{1}{2}a(s)z(s)^2 + v(s)z(s),\ v' \leftarrow v(s) + a(s)z(s)$
  - For the next $s' \in S$, check if $x(s') = x'$ and $v(s') = v'$. If not, INVALID.

After verifying the travel is valid, we get a mapping from location to time, i.e., the functions f and g in the problem definition. Thus we can construct a set $\mathcal{J}$ of jobs where each job $\tau \in \mathcal{J}$ has an execution time $e(\tau)$ and a set of feasible intervals $\mathcal{I}(\tau)$, in which each feasible interval $I \in \mathcal{I}(\tau)$ has a release time $r(I)$ and a deadline $d(I)$.

For $\mathcal{J}$, we verify the job allocation T is valid by the following procedures:

- Initialize $s(\tau) \leftarrow 0$ for all $\tau \in \mathcal{J}$.
- Repeat for all $t \in T$:
  - For job τ with ID $id(t)$, check if $\exists I \in \mathcal{I}(\tau).\ t_s(t) \geq r(I), t_e(t) \leq d(I)$. If not, INVALID.
  - Update $s(\tau) \leftarrow s(\tau) + (t_e(t) - t_s(t))$
- For all $\tau \in \mathcal{J}$, check if $s(\tau) = e(\tau)$. If not, INVALID.

The verification procedures for S and T and the construction of $\mathcal{J}$ are done in time polynomial in the size of S and T. Since S and T are polynomial size in the length of the problem instance, it is a polynomial time verifier for GENERALIZED 1-D DATA MULE SCHEDULING. $\square$

5.2.2 NP-hardness: for fixed $k \geq 2$

Theorem 5.3. *GENERALIZED 1-D DATA MULE SCHEDULING with k feasible location intervals is NP-hard for any fixed $k \geq 2$.*

Proof. 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 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. For simplicity we assume $s(a_i) = a_i$ in the rest of the proof.

Figure 5.2 shows an instance of GENERALIZED 1-D DATA MULE SCHEDULING problem we construct. Each job contains two feasible location intervals. The set $\mathcal{J}_L$ of location jobs consists of following three types. Each location job is denoted as $\{$(set of feasible location intervals), (execution time)$\}$.

- Type-I: Subset choices:

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

- Type-II: Stop points:

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

- Type-III: Equalizers:

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

and $\mathcal{J}_L = \mathcal{J}_{L,1} \cup \mathcal{J}_{L,2} \cup \mathcal{J}_{L,3}$, where

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

and $[z]$ is the short notation for a zero-length feasible location interval $[z, z]$. We set the maximum absolute acceleration $a_{max} = 1$ and the total travel interval $[X_s, X_d] = [0, 2L_n]$. Type-II and III jobs need one padding interval (defined above) for each, but it is omitted from the above definition for clarity.

Intuitively, each of 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. 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 possible movements of the data mule by forcing it to stop at certain locations. A type-II job has one zero-length feasible location interval[3]. To execute the job, the data mule needs to stop at the location. We call these locations "stop points" afterwards.

Figure 5.3 is an excerpt related to a decision on one variable. It shows three possible travels to finish all Type-I and II jobs in these ranges. The data mule must stop at points A, B, C, and D where we have type-II jobs. To execute type-I job (job I-i), the data mule also 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 5.3, the data mule should accelerate in the maximum acceleration ($a_{max} = 1$) first, then at the middle of A and B, decelerate at the negative maximum acceleration ($-a_{max}$). In Figure 5.3, there are three possible distances to make such

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

[3]Each type-II job also has one padding interval.

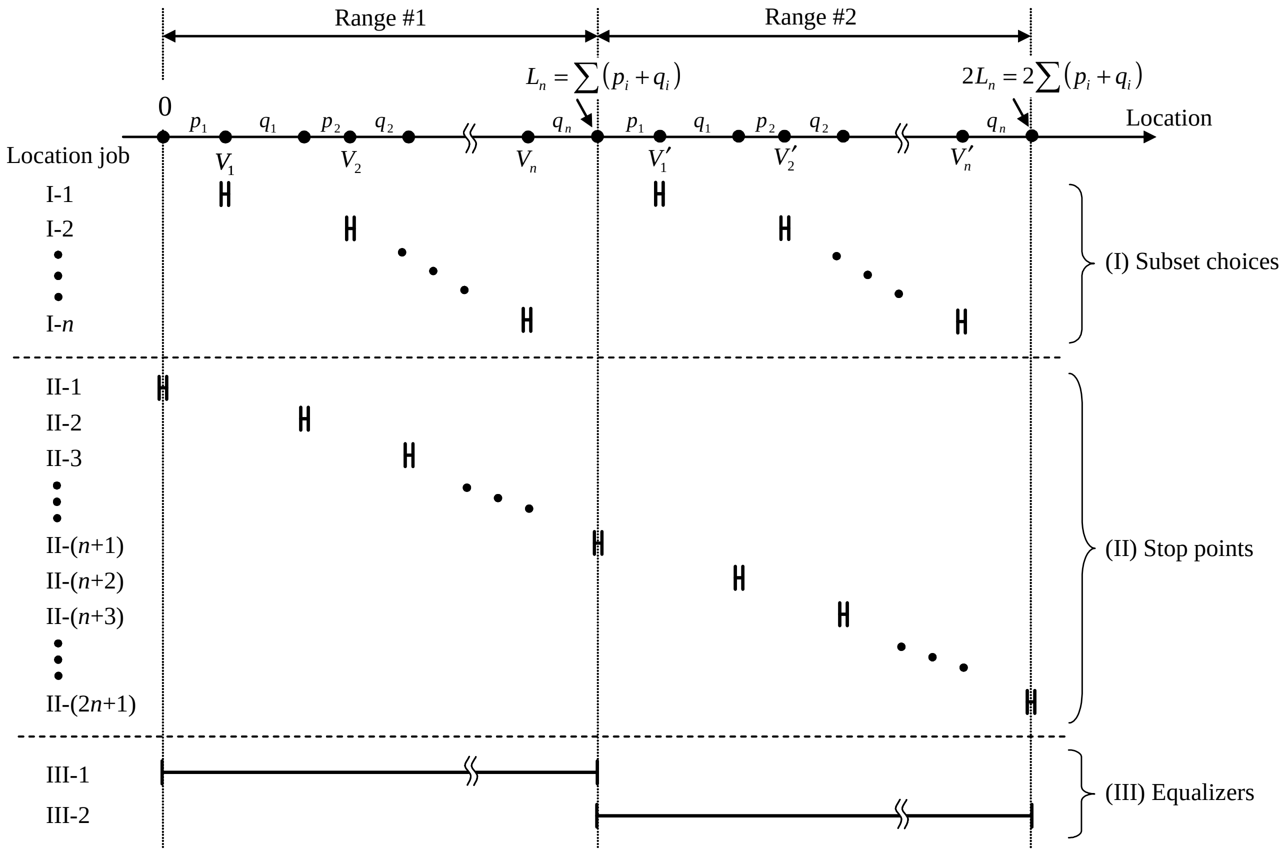


Figure 5.2: Reduction from PARTITION: Each row corresponds to one job. For type-I and type-II jobs, all feasible location intervals are zero-length and the execution time is one. Type-III jobs have feasible location intervals with the length of L_n.

jumps: p_i, q_i, and $(p_i + q_i)$, and we denote the fastest time of these jumps by $t_{S,i}$, $t_{M,i}$, and $t_{L,i}$, respectively. Then we have $p_i = \frac{a_{max}}{4} t_{S,i}^2,\ q_i = \frac{a_{max}}{4} t_{M,i}^2,\ p_i + q_i = \frac{a_{max}}{4} t_{L,i}^2$, and thus

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

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

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

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

Type-III jobs works as equalizers: they force the data mule to spend equal time in each range. Each type-III job has a feasible location interval covering the whole range[4]. 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 $\sum_i t_{L,i} (= \frac{5}{2}S)$ seconds, 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 $\sum_i a_i (= S)$

[4]Each type-III job also has one padding interval at the end of the first feasible location interval.

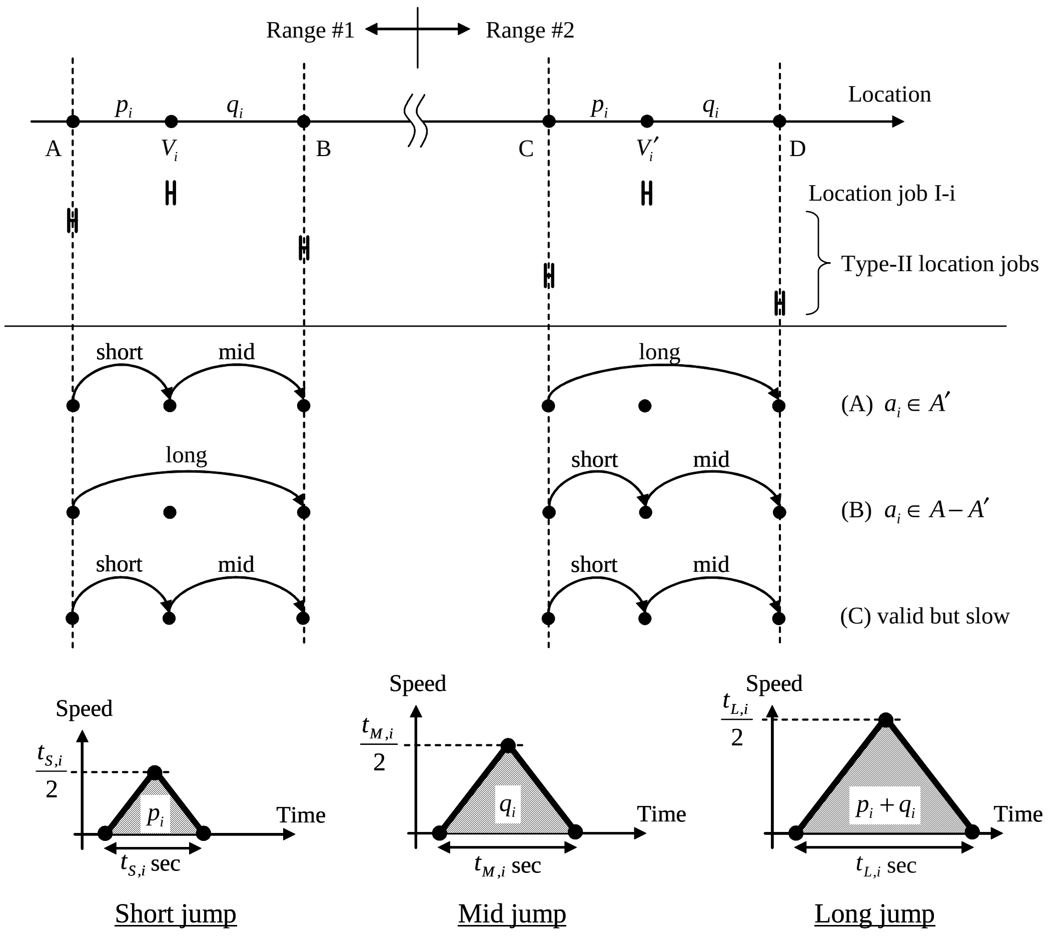


Figure 5.3: Possible travels and corresponding choices for i-th variable. The values of p_i, q_i are determined such that one short jump plus one mid jump ($t_{S,i} + t_{M,i}$) take longer than one long jump ($t_{L,i}$) exactly by a_i seconds, excluding the time the data mule is not moving.

seconds 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 $2\sum_i t_{L,i} + S (= 6S)$ seconds to move from the start to the destination. Each of type-III jobs has the execution time equal to the half of this ($= 3S$), and thereby we make the data mule spends the same time in each range.

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 5.3 and also the possibility of other "wasteful" movements[5].

Finally, since the total travel time $T = T_S + T_M$, we set T to its minimum possible value:

$$T \quad = \quad 3n + 1 + 6S \tag{5.3}$$

The construction above is done in time polynomial to the size of the original PARTITION problem.

($\Rightarrow$) For correctness, we first construct a set of acceleration changing points S and a job allocation schedule, from a satisfying partition for the set A, denoted by $A' \subseteq A$. An acceleration changing point s is represented as a quadruple $\{x(s), z(s), v(s), a(s)\}$, where $x(s) \in \mathbf{Q}_0^+$ is the current location, $z(s) \in \mathbf{Q}_0^+$ is the duration, $v(s) \in \mathbf{Q}_0^+$ is the current speed, and $a(s) \in \mathbf{Q}$ is the acceleration. We prepare a set S_S of acceleration changing

[5]Examples of wasteful movements include: not moving in the maximum possible speed, stopping at locations without zero-length feasible location intervals, not accelerating/decelerating at the maximum acceleration, etc.

points for stopping to execute type-I and type-II jobs as follows. Note the speed and acceleration are all zero for these.

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

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

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

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

where S_{M+} and S_{M-} correspond to the cases (A) and (B) in Figure 5.3, respectively.

Then we construct a set $S = S_S \cup S_M$. S is a valid schedule, since S_M makes a valid movement and assures the data mule to stop at every location included in S_S. 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 set S. Then, since $S_2, S_{1+}, S_{1-}, S_{M+}, S_{M-}$ are exclusive to each other, $||S_2|| = 2n+1,\ ||S_{1+} \cup S_{1-}|| = n,\ ||S_S|| = ||S_2|| + ||S_{1+} \cup S_{1-}|| = 3n+1,\ ||S_M|| = ||S_{M+} \cup S_{M-}|| = 6S$ and thus $||S|| = ||S_S|| + ||S_M|| = 3n+1+6S$. 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 satisfying movement schedule for GENERALIZED 1-D DATA MULE SCHEDULING problem characterized by the set $\mathcal{J}_L$ of location jobs, the maximum absolute acceleration $a_{max} = 1$, the start $X_s = 0$, the destination $X_d = 2L_n$, and the total travel time $T = 3n+1+6S$.

($\Leftarrow$) Next, we construct a satisfying partition for A from a satisfying schedule for GENERALIZED 1-D DATA MULE SCHEDULING problem above. We can immediately construct one as follows:

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

From the discussion on Figure 5.3, the data mule stops and executes the type-I job at either V_i or V_i' $(1 \le i \le n)$ but not both, which makes the above construction valid. In addition, it stops at every stop point and execute

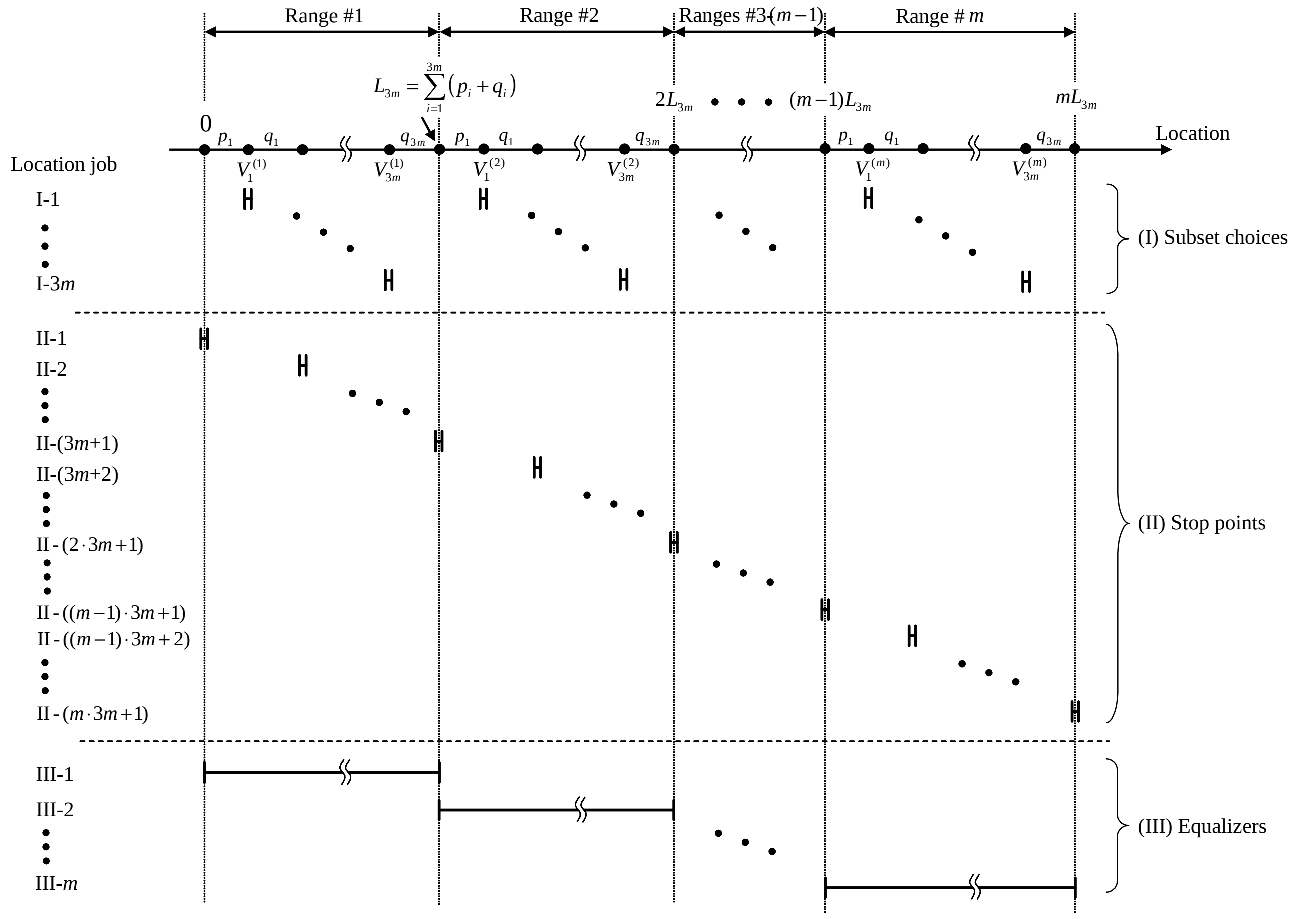


Figure 5.4: Reduction from 3-PARTITION: Each row corresponds to one job. For type-I and type-II jobs, all feasible location intervals are zero-length and the execution time of jobs is one. Type-III jobs have feasible location intervals with the length of L_n.

a type-II job. From the conditions that both of the type-III jobs are completed, the movement of the data mule takes $\frac{S}{2}$ seconds[6] additional 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, it means $\sum_{a_i \in A'} a_i = \sum_{a_i \in A - A'} a_i = S/2$, which is a valid partitioning. □

The following corollary immediately follows:

Corollary 5.4. *GENERALIZED 1-D DATA MULE SCHEDULING with k feasible location intervals is NP-complete for any fixed $k \geq 2$.*

5.2.3 NP-hardness in the strong sense: for k arbitrary

Theorem 5.5. *GENERALIZED 1-D DATA MULE SCHEDULING with k feasible location intervals is NP-hard for k arbitrary.*

Proof. 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 $s(a_i) = a_i$.

[6]Note it only accounts for the moving time, i.e., time to execute type-I jobs at the variable points is excluded.

Figure 5.4 shows an instance of GENERALIZED 1-D DATA MULE SCHEDULING we construct. The location axis is divided into m ranges, each of which is analogous to the one used in the proof for $k \geq 2$ case. The set $\mathcal{J}_L$ of location jobs consists of following three types of job.

- Type-I: Subset choices:

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

- 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:

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

and $\mathcal{J}_L = \mathcal{J}_{L,1} \cup \mathcal{J}_{L,2} \cup \mathcal{J}_{L,3}$, where

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

and $[z]$ is the short notation for a zero-length feasible location interval $[z, z]$. We set the maximum absolute acceleration $a_{max} = 1$ and the total travel range $[X_s, X_d] = [0, mL_{3m}]$. Choices of p_i and q_i are based on exactly same discussion as the proof for $k \geq 2$ case. 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.

Similarly as the $k \geq 2$ case, the fastest travel for each range takes $\sum t_{L,i} = \frac{5}{2} mB$. 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 $k \geq 2$ case. □

The following corollary immediately follows:

Corollary 5.6. *GENERALIZED 1-D DATA MULE SCHEDULING with k feasible location intervals is NP-complete in the strong sense for k arbitrary.*

5.3 Relations with speed scaling problem

As discussed in Chapter 4, we can regard the speed $v(t)$ of data mule corresponds to the inverse of the processor speed $s(t)$ in a special case of dynamic speed scaling (DSS) problem, in which power function $P(s) = s$. When we constrain the acceleration to a_{max}, it corresponds to a constraint on the rate of processor speed change in DSS problem.

There are a few papers on DSS that adopts the assumption of constrained rate of processor speed change. Hong et al. [HQPS98] assumes constant maximum rate of processor speed change (i.e., $|dS(t)/dt| \leq K$, where $S(t)$ is processor speed function and K is a constant). They analyze possible speed changes under this constraint and present some findings about the relation among processor speed, incurred time delay and workload, which are directly applicable to the data mule scheduling problem. For example, Theorem 2 in [HQPS98] corresponds to our discussion on "equivalent movement" appears later (in Section 6.1.2), which we use for formulating the problem as an mathematical optimization problem. In a subsequent paper, Yuan and Qu [YQ05] classify the models of DVS 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. Hong et al.'s work [HQPS98] uses the "optimistic feasible" model and the data mule scheduling problem also corresponds to it.

However, these constraints on the acceleration of data mule and processor speed are not identical. We can describe the acceleration constraint as $|v_1 - v_0| \leq a_{max}t$, where v_0 is the initial speed and v_1 is the speed after t seconds. In the same way, the constraint on processor speed is described as $|s_1 - s_0| \leq Kt$. We can convert it to $|\frac{1}{s_1} - \frac{1}{s_0}| \leq K s_0 s_1 t$, remembering that v_0, v_1 corresponds to $\frac{1}{s_0}, \frac{1}{s_0}$, respectively. This constraint is different from our acceleration constraint, since the maximum rate depends on the start and finishing speed.

Chapter 6

Mathematical Formulation

In this chapter we formulate the general case of data mule scheduling as a quadratic programming (QP) problem. For the QP problem, we also present two ways of finding a lower bound of the optimal solution. One way is by SDP (semidefinite programming) relaxation, and the other is by more problem-specific analysis and only for simple location jobs.

6.1 Approach

We formulate only the speed control subproblem here, as we can formulate the job scheduling subproblem as a linear program, as we saw in 3.3.2. For speed control, we use the feasibility test based on processor demand [BHR93] and formulate the problem of minimizing total travel time as a quadratic program.

6.1.1 Relations between location, time, and speed

Figure 6.1 shows the relationship between time and speed when the acceleration is constrained. Suppose the data mule moves from location l_i to l_{i+1}. At location l_i, time is t_i and the speed is v_i, and at l_{i+1} we have t_{i+1} and v_{i+1}.

Let's focus on Case (a) in the figure. A change of speed over time is expressed as a curve in a time-speed graph. Under the constraint on maximum absolute acceleration, all possible changes of speed over time interval $[t_i, t_{i+1}]$ are confined in the rectangle CFDE. As the area between the curve and the time axis corresponds to the distance, we have the following relationship:

$$\begin{aligned} l_{i+1} - l_i &\geq S(ACEDB) \qquad (6.1)\\ l_{i+1} - l_i &\leq S(ACFDB) \qquad (6.2)\end{aligned}$$

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

$$\begin{aligned} S(ACDB) &= \frac{1}{2}(v_i + v_{i+1})z_i \quad (\equiv \alpha) \qquad (6.3)\\ S(CDE) &= \frac{1}{4}\left\{a_{max}z_i^2 - \frac{(v_{i+1}-v_i)^2}{a_{max}}\right\} \quad (\equiv \beta) \qquad (6.4)\\ S(ACEDB) &= S(ACDB) - S(CDE) = \alpha - \beta \qquad (6.5)\\ S(ACFDB) &= S(ACDB) + S(CFD) = S(ACDB) + S(CDE) = \alpha + \beta \qquad (6.6)\end{aligned}$$

As for Case (b) in Figure 6.1, 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' as in Figure 6.1 is impossible. Instead, the speed change along C'-G'-H'-D' realizes the shortest travel distance, which is equal to $S(A'C'G') + S(H'D'B')$. In this case, in addition to (6.2), there is a following constraint:

$$\begin{aligned} l_{i+1} - l_i &\geq S(A'C'G') + S(H'D'B')\\ &= \frac{v_i^2 + v_{i+1}^2}{2a_{max}} \qquad (6.7)\end{aligned}$$

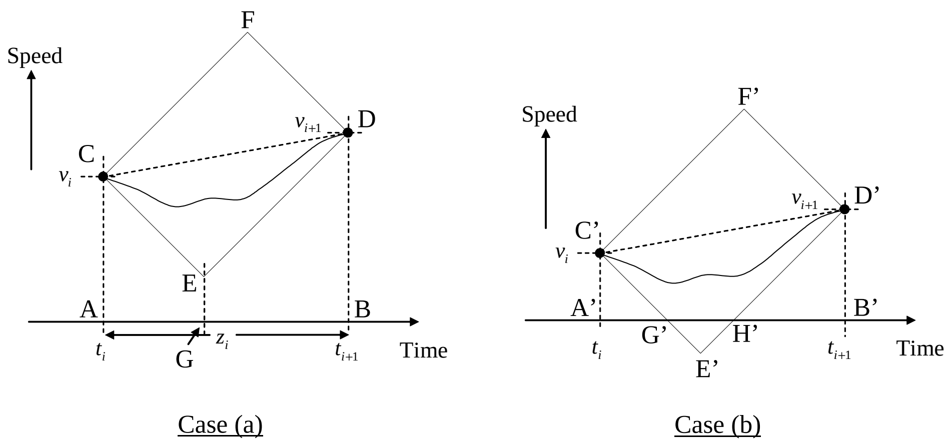


Figure 6.1: Time and speed under constrained acceleration: All possible transitions from point C (t_i, v_i) to D (t_{i+1}, v_{i+1}) are confined in the rectangle CFDE in case (a) or the pentagon C'F'D'H'G' in case (b). The lines CF, ED, C'F', H'D' represent maximum acceleration (i.e., the slope is a_{max}) and CE, FD, C'G', F'D' represent maximum deceleration (i.e., the slope is $-a_{max}$).

Now we consider the constraints for general case. Since the vertical coordinate of point E is

$$\frac{z_i}{2} - \frac{v_{i+1} - v_i}{2a_{max}} \tag{6.8}$$

we can summarize the constraints as follows:

$$\begin{aligned} l_{i+1} - l_i &\geq \alpha - \beta && (6.9) \\ l_{i+1} - l_i &\leq \alpha + \beta && (6.10) \\ l_{i+1} - l_i &\geq \frac{v_i^2 + v_{i+1}^2}{2a_{max}} \quad (\text{if } a_{max} z_i - (v_{i+1} - v_i) \leq 0) && (6.11) \end{aligned}$$

where

$$\begin{aligned} \alpha &= \frac{1}{2}(v_i + v_{i+1}) z_i && (6.12) \\ \beta &= \frac{1}{4}\left\{ a_{max} z_i^2 - \frac{(v_{i+1} - v_i)^2}{a_{max}} \right\} && (6.13) \end{aligned}$$

Note (6.9) is a constraint for both cases, since (6.11) is a stronger constraint than (6.9) for Case (b).

6.1.2 Constructing an equivalent movement

After we obtain a time-speed profile that satisfies the constraints, we need to interpolate this to generate a continuous time-speed change. Specifically, we will connect two points $(t_i, v_i), (t_{i+1}, v_{i+1})$ on the time-speed graph. In addition to these two points, the distance $(l_{i+1} - l_i)$ that the data mule travels in time range $[t_i, t_{i+1}]$ is also given. Recall that the distance is the area between time-speed curve and the time axis. Clearly, there are infinite number of "equivalent" movements that start from (t_i, v_i), end at (t_{i+1}, v_{i+1}), and travel the distance $(l_{i+1} - l_i)$ under the maximum acceleration constraint.

We can construct one of the simplest movements among these equivalent movements as follows. The movement consists of up to three line segments having a slope of either $a_{max}, -a_{max}$ or 0. In addition, the change of the slope of the segments is in the form of "$(\pm a_{max}, 0, \pm a_{max})$", where each of these may be omitted. In other words, the movement changes the acceleration at up to two points and these points have the same speed.

Figure 6.2 shows the movements we construct. Let r_i, s_i denote the time at which we change the acceleration and u_i denote the speed at these points. Depending on the parameters, there are three types of movements.

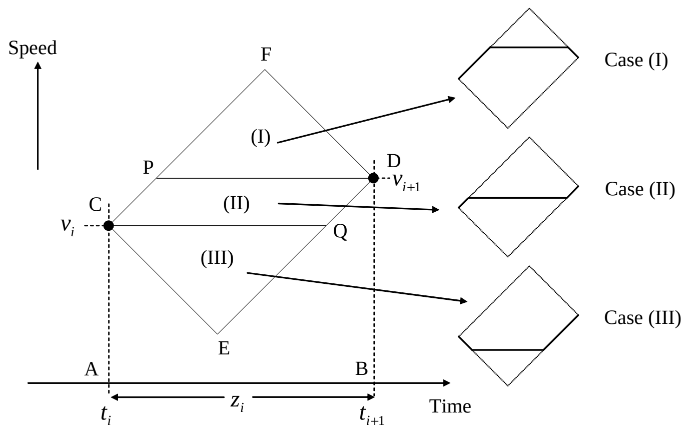


Figure 6.2: Constructing a simple equivalent movement: Depending on the distance $(l_{i+1} - l_i)$ to move between time t_i and t_{i+1}, we can construct a movement in either of three types. Note the constructed movements consist of only maximum acceleration/deceleration and constant speed.

We define p as follows:

$$\begin{aligned} p &= CQ = PD \\ &= z_i - \left| \frac{v_{i+1} - v_i}{a_{max}} \right| \end{aligned} \tag{6.14}$$

The parameters for each equivalent movement are as follows:

- Case (I): If $S(ACPDB) \leq l_{i+1} - l_i \leq S(ACFDB)$

  Acceleration change is Accel-Const-Decel.

$$(r_i, s_i) = \begin{cases} (t_{i+1} - p + q_1,\ t_{i+1} - q_1) & \text{if } v_i \leq v_{i+1} \\ (t_i + q_1,\ t_i + p - q_1) & \text{if } v_i > v_{i+1} \end{cases} \tag{6.15}$$

$$u_i = \max(v_i, v_{i+1}) + a_{max} q_1 \tag{6.16}$$

  where

$$q_1 = \frac{1}{2} \left(p - \sqrt{p^2 - \frac{4((l_{i+1} - l_i) - S(ACPDB))}{a_{max}}} \right) \tag{6.17}$$

- Case (II): If $S(ACQDB) \leq l_{i+1} - l_i < S(ACPDB)$

  Acceleration change is Accel-Const-Accel if $v_i < v_{i+1}$ and Decel-Const-Decel if $v_i > v_{i+1}$. When $v_i = v_{i+1}$, the movement degenerates to constant speed.

$$(r_i, s_i) = \begin{cases} (t_i + q_2,\ t_i + p + q_2) & \text{if } v_i \leq v_{i+1} \\ (t_{i+1} - q_2 - p,\ t_{i+1} - q_2) & \text{if } v_i > v_{i+1} \end{cases} \tag{6.18}$$

$$u_i = \min(v_i, v_{i+1}) + a_{max} q_2 \tag{6.19}$$

  where

$$q_2 = \frac{1}{2} \left(p - \sqrt{p^2 - \frac{4((l_{i+1} - l_i) - S(ACQDB))}{a_{max}}} \right) \tag{6.20}$$

- Case (III): If $S(ACEDB) \leq l_{i+1} - l_i < S(ACQDB)$
  Acceleration change is Decel-Const-Accel.

$$\begin{aligned} (r_i, s_i) &= \begin{cases} (t_i + q_3,\ t_i + p - q_3) & \text{if } v_i \leq v_{i+1} \\ (t_{i+1} - p + q_3,\ t_{i+1} - q_3) & \text{if } v_i > v_{i+1} \end{cases} && (6.21) \\ u_i &= \min(v_i, v_{i+1}) - a_{max} q_3 && (6.22) \end{aligned}$$

where

$$q_3 = \frac{1}{2}\left(p - \sqrt{p^2 - \frac{4(S(ACQDB) - (l_{i+1} - l_i))}{a_{max}}}\right) \tag{6.23}$$

6.2 Quadratic programming formulation

Variables For each location l_i $(i = 0, ..., 2m + 1)$,

- v_i: speed of data mule

and for each location interval $[l_i, l_{i+1}]$ $(i = 0, ..., 2m)$,

- z_i: time to stay in the interval[1]
- $p_i(\tau_L)$: time allocated to location job τ_L

Objective Minimize

$$\sum_{i=0}^{2m} z_i \tag{6.24}$$

Constraints

- (One-way movement)

$$\begin{aligned} z_i &\geq 0 && (6.25) \\ v_i &\geq 0 && (6.26) \end{aligned}$$

- (Job completion) For all $\tau_L \in \mathcal{J}_L$

$$\sum_{i=0}^{2m+1} p_i(\tau_L) = e(\tau_L) \tag{6.27}$$

- (Feasible interval) For all $\tau_L \in \mathcal{J}_L$, if $\forall I \in \mathcal{I}(\tau_L), [l_i, l_{i+1}] \notin I$,

$$p_i(\tau_L) = 0 \tag{6.28}$$

- (Processor demand)

$$\sum_{\tau_L \in \mathcal{J}_L} p_i(\tau_L) \leq z_i \tag{6.29}$$

[1]Strictly speaking, this interpretation is not appropriate when $l_i = l_{i+1}$, but the formulation remains valid.

- (Maximum absolute acceleration)

$$\alpha - \beta \leq l_{i+1} - l_i \leq \alpha + \beta \tag{6.30}$$

$$\frac{v_i^2 + v_{i+1}^2}{2a_{max}} \leq l_{i+1} - l_i \quad (\text{if } a_{max}z_i - (v_{i+1} - v_i) \leq 0) \tag{6.31}$$

$$|v_{i+1} - v_i| \leq a_{max}z_i \tag{6.32}$$

where

$$\alpha = \frac{1}{2}(v_i + v_{i+1})z_i \tag{6.33}$$

$$\beta = \frac{1}{4}\left\{a_{max}z_i^2 - \frac{(v_{i+1} - v_i)^2}{a_{max}}\right\} \tag{6.34}$$

The constraint (6.31) 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:

$$b_i(1 - b_i) = 0 \tag{6.35}$$

$$b_i(a_{max}z_i - (v_{i+1} - v_i)) \geq 0 \tag{6.36}$$

$$(1 - b_i)(a_{max}z_i - (v_{i+1} - v_i)) \leq 0 \tag{6.37}$$

$$(1 - b_i)(w_i - 2a_{max}(l_{i+1} - l_i)) \leq 0 \tag{6.38}$$

$$v_{i+1}^2 + v_i^2 = w_i \tag{6.39}$$

6.3 Finding lower bounds

We presented a formulation of data mule scheduling problem in the previous section. Since it is a nonconvex quadratic program (QP), global optimization is often hard even for a small size input. In this section we present two methods to find a lower bound of the global optimal solution. One is by SDP (semidefinite programming) relaxation, which is a generic method for nonconvex QP problems. The other is only for simple location jobs and exploits the characteristics of the problem by calculating the maximum possible speed at each location.

6.3.1 SDP (semidefinite programming) relaxation

We construct a SDP relaxation problem to this nonconvex QP problem. The domain of a relaxation problem contains that of the original problem, so the minimum value in the relaxation problem gives the lower bound of the original problem, assuming it is a minimization problem. We first omit constraint (6.31) for simplification and then apply SDP relaxation. SDP is convex programming and thus solved efficiently (see [BV04] etc.).

Consider the following quadratic programming problem:

$$\begin{array}{ll} \text{minimize} & x^T A_0 x + b_0^T x + c_0 \\ \text{subject to} & x^T A_i x + b_i^T x + c_i \leq 0 \quad (i = 1, ..., m) \end{array} \tag{6.40}$$

with a variable $x \in \mathbf{R^n}$ and parameters $A_i \in \mathbf{S}^n$, $b_i \in \mathbf{R}^n$, and $c_i \in \mathbf{R}$ for $i = 1, ..., m$, where $\mathbf{S}^n, \mathbf{R}^n, \mathbf{R}$ are the sets of symmetric $n \times n$ matrices, real n-vectors, and real numbers, respectively. When $A_i \succeq 0$ for all i, i.e., all A_i's are positive semidefinite, (6.40) is semidefinite program and can be solved efficiently.

In SDP relaxation, we introduce a new variable $X \in \mathbf{S}^n$ to replace quadratic terms $x_i x_j$. Then the problem (6.40) is rewritten as an equivalent QP problem as follows:

$$\begin{array}{ll} \text{minimize} & A_0 \cdot X + b_0^T x + c_0 \\ \text{subject to} & A_i \cdot X + b_i^T x + c_i \leq 0 \quad (i = 1, ..., m) \\ & X = xx^T \end{array} \tag{6.41}$$

where $A \cdot B \equiv \sum_{i,j} A_{ij} B_{ij}$. SDP relaxation replaces the equality constraint $X = xx^T$ by a positive semidefiniteness constraint $X - xx^T \succeq 0$. Further, by using the Schur complement, we obtain the following SDP problem that is a relaxation of the original QP problem:

$$\begin{array}{rl} \text{minimize} & A_0 \cdot X + b_0^T x + c_0 \\ \text{subject to} & A_i \cdot X + b_i^T x + c_i \leq 0 \quad (i = 1, ..., m) \\ & \begin{bmatrix} X & x \\ x^T & 1 \end{bmatrix} \succeq 0 \end{array} \tag{6.42}$$

In our formulation, (6.30) is the only quadratic constraint after we omit (6.31). We introduce a matrix variable $X \equiv xx^T$ where $x = [v^T | z^T]^T$ and replace (6.30) with linear constraints as follows:

$$\alpha' - \beta' \quad \leq \quad l_{i+1} - l_i \ \leq \ \alpha' + \beta' \tag{6.43}$$

where

$$\alpha' \quad = \quad \frac{1}{2}(X_{i,n+i} + X_{i+1,n+i}) \tag{6.44}$$

$$\beta' \quad = \quad \frac{1}{4}\left\{a_{max} X_{n+i,n+i} - \frac{1}{2}(X_{i+1,i+1} - 2X_{i,i+1} + X_{i,i})\right\} \tag{6.45}$$

Then we add the positive semidefiniteness constraint:

$$\begin{bmatrix} X & x \\ x^T & 1 \end{bmatrix} \quad \succeq \quad 0 \tag{6.46}$$

6.3.2 Another lower bound for simple location jobs

For simple location jobs having only one feasible location interval for each, we can obtain another lower bound in a different way by exploiting the properties of the problem. The main idea of the procedure, which we call LB-MaxSpeed procedure, is to calculate the maximum possible speed at each release or deadline locations (denoted l_i in the previous formulations) and find the fastest possible travel that satisfies these speed constraints.

LB-MaxSpeed procedure finds the minimum travel time that satisfies both of the following two constraints:

1. Maximum speed constraint at each l_i
2. Minimum time constraint for each interval $[l_i, l_j]$

Both of them are based on processor demand analysis.

The first constraint is about the maximum speed at each l_i. For each location interval consisting of a release location and a deadline location, processor demand determines the maximum possible speed within the interval. Maximum speed at l_i is upper bounded by the above maximum speed for each interval. The details of the procedure is described by the following pseudocode. In lines 4-6, v_b is the maximum possible speed at the edge of the location interval I, assuming the constant acceleration at a_{max} within the interval. Line 6 corresponds to the case when the constant acceleration within I is not possible because the speed must always be nonnegative. Lines 8-11 calculate an upper bound of the speed at l_i when the maximum speed at the edge of I is v_b. In Line 12, the maximum speed at each l_i is determined by the lowest of all the upper bounds.

```
1   for each $l_i$
2       do for each location interval $I = [r(\tau), d(\tau')]$ s.t. $\tau'_L, \tau''_L \in \mathcal{J}_L, r(\tau'_L) \leq d(\tau''_L)$
3              do $g(I) \leftarrow$ Processor-Demand$(\mathcal{J}_L, I)$
4                  if $\frac{|I|}{g(I)} - \frac{a_M g(I)}{2} > 0$
5                     then $v_b \leftarrow \frac{|I|}{g(I)} + \frac{a_M g(I)}{2}$
6                     else $v_b \leftarrow \sqrt{2 a_M g(I)}$
7                  if $l_i \notin I$
8                     then $d \leftarrow \max\{|low[I] - l_i|, |high[I] - l_i|\}$
9                          $v_i[I] \leftarrow \sqrt{v_b^2 + 2 a_M d}$
10                    else $d \leftarrow \min\{l_i - low[I], high[I] - l_i\}$
11                         $v_i[I] \leftarrow \sqrt{\max\{0, v_b^2 - 2 a_M d\}}$
12         $v_i \leftarrow \min_I v_i[I]$
```

The second constraint is about the minimum time for each interval $[l_i, l_j](i < j)$, which is determined by the processor demand for the interval. By using variables z_i to represent the time to stay in the interval $[l_i, l_{i+1}]$, we can obtain a set of linear constraints.

From the first constraint, we can obtain the lower bound of the time the data mule takes to travel each interval $[l_i, l_{i+1}]$. Then we can combine these two sets of constraints to formulate the problem as a linear program as follows:

Variables

- z_i: time to stay in the location interval $[l_i, l_{i+1}]$

Objective Minimize

$$\sum_{i=0}^{2m} z_i \tag{6.47}$$

Constraints

- (Maximum speed)

$$z_i \geq \frac{2}{a_M}\sqrt{a_M(l_{i+1} - l_i) + \frac{v_i^2 + v_{i+1}^2}{2}} - \frac{v_i + v_{i+1}}{a_M} \tag{6.48}$$

  The right hand side is the minimum travel time starting from l_i at the speed v_i and finishing at l_{i+1} at the speed v_{i+1}. This is a linear constraint since there is no variable on the right hand side.

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

$$\sum_{\tau_L \in \mathcal{J}_L, I(\tau_L) \in I} e(\tau_L) \leq \sum_{[l_i, l_{i+1}] \in I} z_i \tag{6.49}$$

Chapter 7

Heuristic Algorithm

In the previous chapter, we presented a formulation of GENERALIZED 1-D DATA MULE SCHEDULING as a quadratic programming problem. Since it is not a convex optimization, finding the global optimum is hard in general and thus not realistic. In this chapter, we present an alternative approach. We design a heuristic algorithm that gives a good solution in a reasonable amount of time. We also analyze the algorithm on its computational complexity and approximation ratio.

7.1 Approach

7.1.1 Overview

Figure 7.1 shows the idea of the heuristic algorithm[1]. The algorithm works recursively, and in each recursion, we confine ourselves to the following 3-phase speed changing 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 the middle of Figure 7.1, the main idea of the algorithm is to maximize the plateau speed until we have a *tight interval*, which is defined as an interval whose length (in time) is equal to the processor demand for that 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", too.

For the intervals in the plateau interval but not in the tight interval, we can still increase the speed without destroying the feasibility. As shown in the bottom of Figure 7.1, we recursively apply the maximization procedure to them until there is no such region.

7.1.2 Structure

The heuristic algorithm consists of following four steps:

Step 1: Simplify Convert 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 feasible location intervals of a general location job are zero-length, we just distribute the execution time equally to them.

Step 2: Maximize Find the maximum plateau speed and a tight interval. For each interval of a release location and a deadline location, calculate the plateau speed that makes it a tight interval. Maximum plateau speed is the minimum of these plateau speeds for all intervals. From Theorem 3.4, this procedure does not

[1]We design the heuristic algorithm for the case where the speed is constrained to be zero at X_s and X_d, but we can also use it for an unconstrained case by considering a hypothetical travel interval, as discussed later.

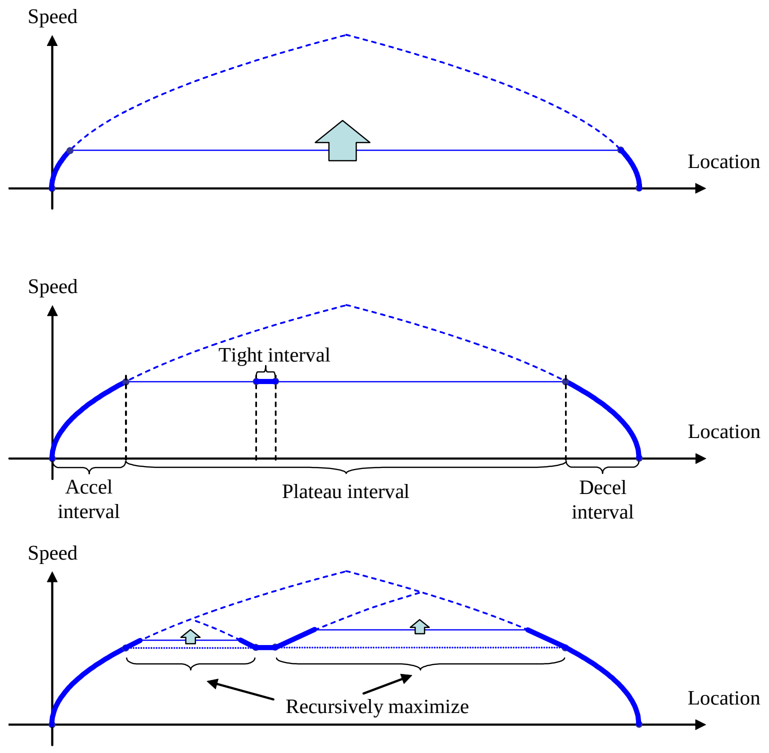


Figure 7.1: The idea of the heuristic algorithm: (Top) Increase the plateau speed. Two curves show the acceleration/deceleration for the fastest possible travel covering the whole interval. Bold lines mean the speed for the interval is already maximized. (Middle) A tight interval found. This is the maximum plateau speed, since we cannot increase it due to this tight interval. Feasible location intervals of the jobs are trimmed so that they don't contain the tight interval as well as accel/decel intervals. (Bottom) Recursively maximize the plateau speed for the remaining unconstrained intervals. Repeat the recursion until there's no such interval.

destroy the feasibility, since the processor demand is equal or less than the time given for each location interval of a release location and a deadline location.

Step 3: Trim Trim feasible location intervals of each (simple) location job. We first process the accel interval and the decel interval, and then the tight interval.

Figure 7.2 explains the details of trimming. For the accel interval, we simulate EDF algorithm to see how much time is allocated to each job within the interval. Let a_i denote the allocated time to job τ_i. A job completely contained in the interval (τ_1) will be completed in the interval (i.e., $a_i = \tau_i$). For a job intersects with the interval (τ_2), we trim off the feasible location interval at the edge of the accel interval, and decrease the execution time to $e_2 - a_2$. For the decel interval, it works in the same way except we simulate LRT (latest release time) algorithm instead of EDF. LRT algorithm schedules jobs backwards, treating release times as deadlines, and is also known to be optimal [Liu00].

The tight interval is processed in a little different way. In the tight interval, by its definition, time is allocated only to the jobs that are completely contained in the interval. Thus, for the jobs that intersect the 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 EDF for the location interval from the end of the accel interval to the beginning of the tight interval[2]. Assuming

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

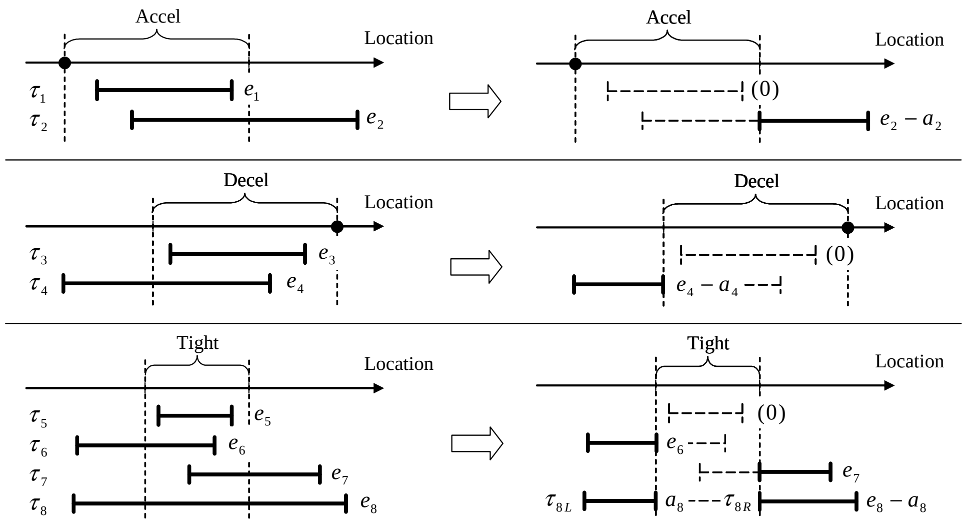


Figure 7.2: Trimming feasible location intervals

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}.

Step 4: Recursion By the previous step, all remaining jobs are separated into two groups. One group of jobs populates the location interval from the end of accel interval to the beginning of the tight interval, and the other populates the location interval from the end of the tight interval to the beginning of the decel interval. The speed for these intervals are not fixed yet, i.e., there may still be some room for increasing the speed without destroying the feasibility. Thus, we recursively maximize the speed by repeating from Step 2 for these intervals. The number of recursions varies from zero to two depending on the configuration of the tight interval.

7.2 Algorithm

The procedure Approx-MotionPlan is the main routine of the algorithm.

```
Approx-MotionPlan($\mathcal{J}, X_s, X_d$)
1  $\mathcal{J}' \leftarrow$ Simplify-Jobset($\mathcal{J}$)
2  return Maximize-Plateau-Speed($\mathcal{J}', [X_s, X_d], 0$)
```

Given a set $\mathcal{J}$ of location jobs[3], the procedure Simplify-Jobset splits each general location job (i.e., with multiple feasible location intervals) into multiple simple location jobs, and return a new set of location jobs. Execution time of new jobs is determined proportionally to the length of their feasible location intervals, so that the total execution time of all the jobs equals to the original general location job. In case a location job only has zero-length feasible location intervals, the execution time is equally distributed to each of them.

[3]For simplicity, we omit the subscript "L" for location jobs and location intervals afterwards.

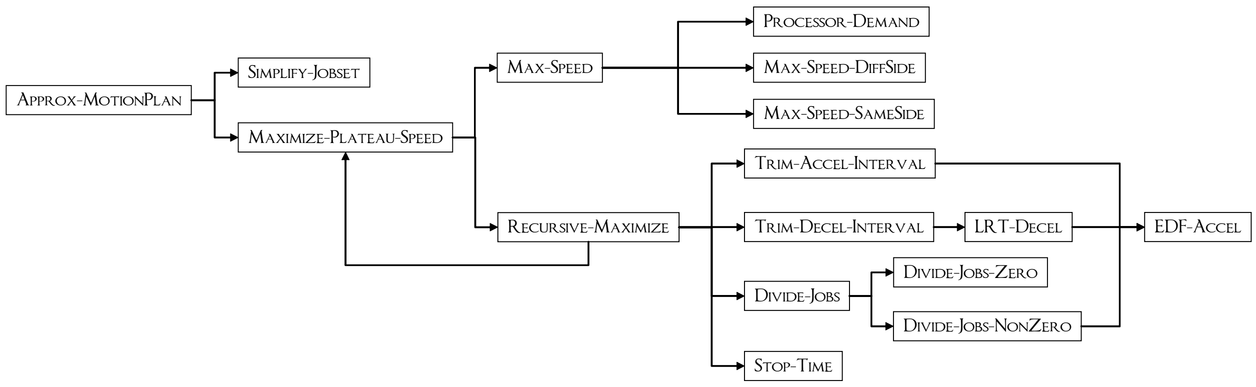


Figure 7.3: Call graph of procedures

```
Simplify-Jobset($\mathcal{J}$)
1  ▷ Init: $\mathcal{J}' \leftarrow \emptyset$
2  for each location job $\tau \in \mathcal{J}$
3        do if $\sum_{I \in \mathcal{I}(\tau)} |I| > 0$
4              then for each feasible location interval $I \in \mathcal{I}(\tau)$
5                        do $\mathcal{J}' \leftarrow \mathcal{J}' \cup \left\{ I, \frac{|I|}{\sum_{I \in \mathcal{I}(\tau)} |I|} e(\tau) \right\}$
6              else  for each feasible location interval $I \in \mathcal{I}(\tau)$
7                        do $\mathcal{J}' \leftarrow \mathcal{J}' \cup \left\{ I, \frac{1}{N(\tau)} e(\tau) \right\}$   ▷ $N(\tau) = \#(\tau$'s feasible location intervals)
8  return $\mathcal{J}'$
```

Given a set $\mathcal{J}$ of location jobs, a baseline interval I_0, and a baseline speed v_0, the procedure Maximize-Plateau-Speed recursively maximizes plateau speed. It returns a list of acceleration changing points, each of which consists of four elements: location interval, initial speed, acceleration, and time duration until the next acceleration changing point.

```
Maximize-Plateau-Speed($\mathcal{J}, I_0, v_0$)
 1  for each location interval $I = [r(\tau), d(\tau')]$ s.t. $\tau, \tau' \in \mathcal{J}$, $r(\tau) \leq d(\tau')$
 2      do $u[I] \leftarrow$ Max-Speed($\mathcal{J}, I$)
 3  $v_p \leftarrow \min_I \{u[I]\}$          ▷ Maximum plateau speed
 4  if $v_p \neq +\infty$
 5      then $I_t \leftarrow \arg\min_I (u[I])$    (If multiple, arbitrarily pick one)      ▷ Tight interval
 6      else  $I_t \leftarrow \emptyset$
 7  $I_p \leftarrow [low[I_0] + (v_p^2 - v_0^2)/2a_{max},\ high[I_0] - (v_p^2 - v_0^2)/2a_{max}]$        ▷ Plateau interval
 8  $I_a \leftarrow [low[I_0], low[I_p]]$                                                   ▷ Accel interval
 9  $I_d \leftarrow [high[I_p], high[I_0]]$                                                 ▷ Decel interval
10  return Recursive-Maximize($\mathcal{J}, I_0, I_a, I_d, I_t, v_0, v_p$)
```

7.2.1 Identifying the tight interval

Given a set $\mathcal{J}$ of location jobs and an interval I, the procedure Max-Speed returns the maximum plateau speed that $\mathcal{J}$ remains feasible in the interval. In other words, Max-Speed returns the speed that makes I a tight interval.

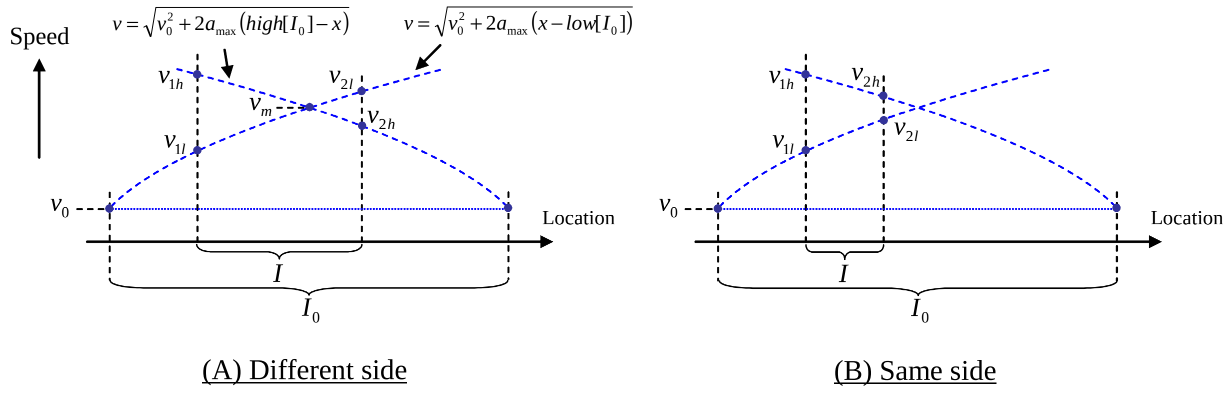


Figure 7.4: Notation in MAX-SPEED

```
MAX-SPEED($\mathcal{J}, I, I_0, v_0$)
 1  $g \leftarrow$ PROCESSOR-DEMAND($\mathcal{J}, I$)
 2  $v_{1l} \leftarrow \sqrt{v_0^2 + 2a_{max}(low[I] - low[I_0])}$
 3  $v_{1h} \leftarrow \sqrt{v_0^2 + 2a_{max}(high[I_0] - low[I])}$
 4  $v_{2l} \leftarrow \sqrt{v_0^2 + 2a_{max}(high[I] - low[I_0])}$
 5  $v_{2h} \leftarrow \sqrt{v_0^2 + 2a_{max}(high[I_0] - high[I])}$
 6  $v_1 \leftarrow \min\{v_{1l}, v_{1h}\}$, $v_2 \leftarrow \min\{v_{2l}, v_{2h}\}$
 7  $v_a \leftarrow \min\{v_1, v_2\}$, $v_b \leftarrow \max\{v_1, v_2\}$
 8  $x_m \leftarrow (low[I_0] + high[I_0])/2$
 9  if $x_m \in I$
10      then return MAX-SPEED-DIFFSIDE($g, I, v_0, v_a, v_b$)
11      else  return MAX-SPEED-SAMESIDE($g, I, v_0, v_a, v_b$)
```

Given a set $\mathcal{J}$ of location jobs and an interval I, the procedure PROCESSOR-DEMAND returns the processor demand in the interval.

```
PROCESSOR-DEMAND($\mathcal{J}, I$)
1  ▷ Init: $d \leftarrow 0$
2  for each location job $\tau \in \mathcal{J}$
3       do if $I(\tau) \in I$        ▷ Note each $\tau$ is a simple location job
4            then $d \leftarrow d + e(\tau)$
5  return $d$
```

The procedures MAX-SPEED-DIFFSIDE and MAX-SPEED-SAMESIDE determine the maximum possible plateau speed to preserve the feasibility condition for an input interval I. As shown in Figure 7.5, depending on the processor demand g, maximum possible speed v_a and v_b at the edge of the interval I, and the baseline speed v_0, we can classify the configuration into four (when $x_m \in I$) or three (when $x_m \notin I$) cases.

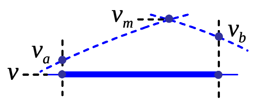
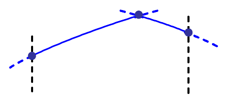
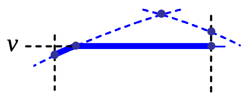
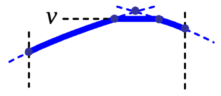
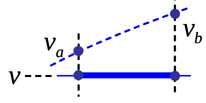
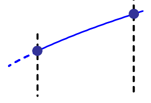
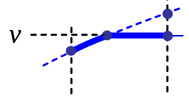
	Case 1	Case 2	Case 3	Case 4
(A) Different side				
(B) Same side				
Max plateau speed	$v = \frac{\lvert I \rvert}{g}$ (g : processor demand for interval I)	$v = +\infty$	$v = c_1 - \sqrt{c_1^2 - c_2}$ $c_1 = v_a + a_{\max} \cdot g$ $c_2 = 2a_{\max} \cdot \lvert I \rvert + v_a^2$	$v = \frac{1}{2}\left(c_3 - \sqrt{c_3^2 - 2c_4}\right)$ $c_3 = (v_a + v_b) + a_{\max} \cdot g$ $c_4 = 2a_{\max} \cdot \lvert I \rvert + (v_a^2 + v_b^2)$

Figure 7.5: Possible cases for plateau speed in (A) Max-Speed-Diffside and (B) Max-Speed-Sameside. Same equations hold for mirror-reversed cases.

Max-Speed-DiffSide(g, I, v_0, v_a, v_b)

```
$t_1 \leftarrow |I|/v_a$
if $g > t_1$
    then return $|I|/g$                              ▷ Case 1
$v_m \leftarrow \sqrt{v_0^2 + a_{max}|I|}$
$t_2 \leftarrow (2v_m - v_a - v_b)/a_{max}$
if $g \leq t_2$
    then return $+\infty$                            ▷ Case 2: No constraint
$t_3 \leftarrow |I|/v_b + (v_b - v_a)^2/2a_{max}v_b$
if $g > t_3$
    then $c_1 \leftarrow v_a + a_{max} \cdot g$
         $c_2 \leftarrow 2a_{max}|I| + v_a^2$
         return $c_1 - \sqrt{c_1^2 - c_2}$              ▷ Case 3
    else $c_3 \leftarrow (v_a + v_b) + a_{max} \cdot g$
         $c_4 \leftarrow 2a_{max}|I| + (v_a^2 + v_b^2)$
         return $(c_3 - \sqrt{c_3^2 - 2c_4})/2$         ▷ Case 4
```

Max-Speed-SameSide(g, I, v_0, v_a, v_b)

```
$t_1 \leftarrow |I|/v_a$
if $g > t_1$
    then return $|I|/g$              ▷ Case 1
$t_2 \leftarrow (v_b - v_a)/a_{max}$
if $g \leq t_2$
    then return $+\infty$            ▷ Case 2
$c_1 \leftarrow v_a + a_{max} \cdot g$
$c_2 \leftarrow 2a_{max}|I| + v_a^2$
return $c_1 - \sqrt{c_1^2 - c_2}$       ▷ Case 3
```

7.2.2 Recursive maximization

After maximizing the plateau speed by identifying a tight interval, we further attempt to increase the speed for the free intervals in the original interval. We recursively perform the maximization procedure for these free intervals until there are no such intervals.

Given a set $\mathcal{J}$ of location jobs, a baseline interval I_0, accel/decel intervals I_a, I_d, a tight interval I_t, baseline speed v_0, plateau speed v_p, and stop time t_s, the procedure Recursive-Maximize returns a list of acceleration changing points. Internally, the procedure identifies the free intervals and then recursively calls MAXIMIZE-PLATEAU-SPEED for them. As shown in Figure 7.6, there are five possible configurations of free intervals and the number of recursive calls varies from zero to two depending on them.

```
RECURSIVE-MAXIMIZE($\mathcal{J}, I_0, I_a, I_d, I_p, I_t, v_0, v_p$)
1  $t_f \leftarrow \dfrac{\sqrt{v_0^2 + 2a_{max}|I_a|} - v_0}{a_{max}}$
2  $s_1 \leftarrow \{I_a, v_0, a_{max}, t_f\}$, $s_2 \leftarrow \{I_d, v_p, -a_{max}, t_f\}$
   $\triangleright$ No recursion
3  if $I_t = \emptyset$                          $\triangleright$ Case 0-1
4     then return $\{s_1, s_2\}$
5  elseif $I_p \in I_t$                          $\triangleright$ Case 0-2
6     then return $\{s_1, \{I_p, v_p, 0, |I_p|/v_p\}, s_2\}$
   $\triangleright$ One recursion
7  $\mathcal{J}_A \leftarrow$ TRIM-ACCEL-INTERVAL($\mathcal{J}, [low[I_a], \min\{high[I_a], low[I_t]\}], v_0$)
8  $\mathcal{J}_{AD} \leftarrow$ TRIM-DECEL-INTERVAL($\mathcal{J}_A, [\max\{low[I_d], high[I_t]\}, high[I_d]], v_0$)
9  if $I_a \cap I_t \neq \emptyset$ and $I_d \cap I_t = \emptyset$          $\triangleright$ Case 1-1
10    then $I_f \leftarrow [high[I_a], high[I_t]]$, $I_r \leftarrow [high[I_t], low[I_d]]$
11          return {  $s_1$,
                      $\{I_f, v_p, 0, |I_f|/v_p\}$,
                      MAXIMIZE-PLATEAU-SPEED($\mathcal{J}_{AD}, I_r, v_p$),
                      $s_2$  }
12 elseif $I_a \cap I_t = \emptyset$ and $I_d \cap I_t \neq \emptyset$     $\triangleright$ Case 1-2
13    then $I_f \leftarrow [low[I_t], low[I_d]]$, $I_r \leftarrow [high[I_a], low[I_t]]$
14          return {  $s_1$,
                      MAXIMIZE-PLATEAU-SPEED($\mathcal{J}_{AD}, I_r, v_p$),
                      $\{I_f, v_p, 0, |I_f|/v_p\}$,
                      $s_2$  }
   $\triangleright$ Two recursions
15 if $I_a \cap I_t \neq \emptyset$ and $I_d \cap I_t \neq \emptyset$          $\triangleright$ Case 2
16    then $I_r \leftarrow [high[I_a], low[I_t]]$, $I_{r'} \leftarrow [high[I_t], low[I_d]]$
17          $\{\mathcal{J}_r, \mathcal{J}_{r'}\} \leftarrow$ DIVIDE-JOBS($\mathcal{J}_{AD}, I_r, I_t, v_p$)
18          if $v_p = 0$
19             then $t_s \leftarrow$ STOP-TIME($\mathcal{J}_{AD}, I_t$)
20             else $t_s \leftarrow |I_t|/v_p$
21          return {  $s_1$,
                      MAXIMIZE-PLATEAU-SPEED($\mathcal{J}_r, I_r, v_p$),
                      $\{I_t, v_p, 0, t_s\}$,
                      MAXIMIZE-PLATEAU-SPEED($\mathcal{J}_{r'}, I_{r'}, v_p$),
                      $s_2$  }
22    else  $\triangleright$ Unreachable
```

The procedure TRIM-ACCEL-INTERVAL allocates the accel interval I_a to the available jobs. It then changes the feasible location intervals of the jobs so that none of them overlaps the accel interval, and returns the modified set of location jobs. Job allocation is based on EDF algorithm.

```
TRIM-ACCEL-INTERVAL($\mathcal{J}, I_a, v_0$)
1  $\triangleright$ Init: $\mathcal{J}_A \leftarrow \emptyset$
2  $a_A \leftarrow$ EDF-ACCEL($\mathcal{J}, I_a, v_0, a_{max}$)          $\triangleright$ Simulate EDF. $a_A$: list of allocated time in $I_a$
3  for each location job $\tau \in \mathcal{J}$
4        do $\tau' \leftarrow \{[\max\{r(\tau), high[I_a]\}, d(\tau)], e(\tau) - a_A(\tau)\}$     $\triangleright$ Trim
5           if $e(\tau') > 0$
6              then $\mathcal{J}_A \leftarrow \mathcal{J}_A \cup \tau'$
7  return $\mathcal{J}_A$
```

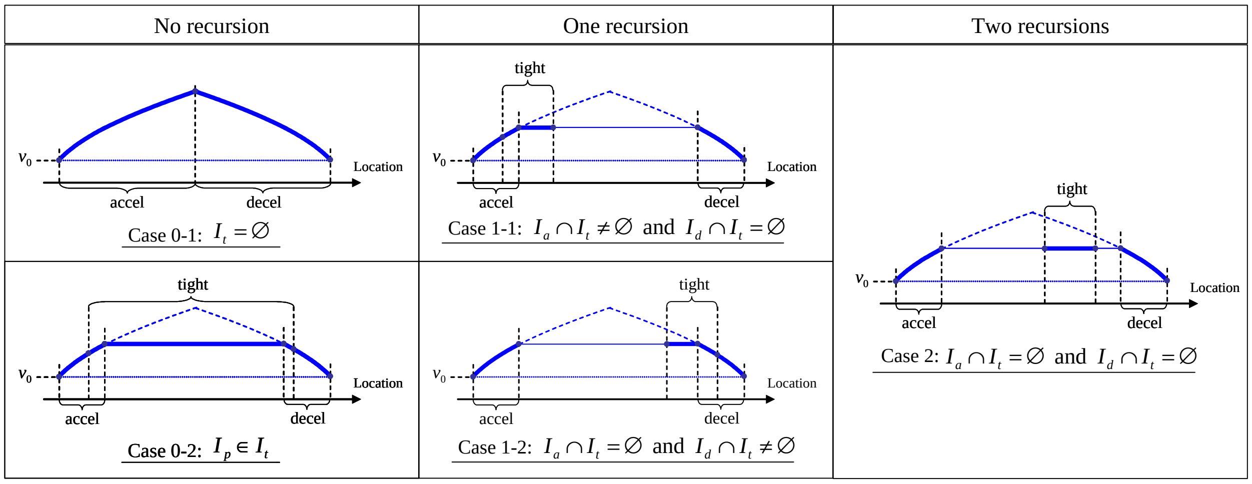

No recursion
One recursion
Two recursions
v_0
Location
accel
decel
tight
Case 0-1: $I_t = \emptyset$
Case 1-1: $I_a \cap I_t \neq \emptyset$ and $I_d \cap I_t = \emptyset$
Case 0-2: $I_p \in I_t$
Case 1-2: $I_a \cap I_t = \emptyset$ and $I_d \cap I_t \neq \emptyset$
Case 2: $I_a \cap I_t = \emptyset$ and $I_d \cap I_t = \emptyset$

Figure 7.6: Possible cases in Recursive-Maximize

Similarly, the procedure Trim-Decel-Interval allocates the decel interval I_d to the available jobs. However, allocation is based on LRT (latest release time) algorithm [Liu00]. LRT algorithm, or "reverse EDF algorithm", schedules the jobs backwards from the end of the interval in priority-driven manner. Contrary to EDF, priorities are based on the release times of jobs.

```
Trim-Decel-Interval($\mathcal{J}, I_d, v_0$)
1  $a_D \leftarrow$ LRT-Decel($\mathcal{J}, I_d, v_0, a_{max}$)    ▷ Simulate LRT. $a_D$: list of allocated time in $I_d$
2  for each location job $\tau \in \mathcal{J}$
3        do $\tau' \leftarrow \{[r(\tau), \min\{d(\tau), low[I_d]\}], e(\tau) - a_D(\tau)\}$    ▷ Trim
4           if $e(\tau') > 0$
5              then $\mathcal{J}_D \leftarrow \mathcal{J}_D \cup \tau'$
6  return $\mathcal{J}_D$
```

The procedure EDF-Accel applies EDF algorithm for the set $\mathcal{J}$ of location jobs within an interval I, assuming the data mule moves at the speed v_0 at the start of the interval and accelerates at a toward the end of the interval. It returns the list of time duration that each job was allocated within the interval.

```
EDF-Accel($\mathcal{J}, I, v_0, a$)
 1  ▷ Init: $a(\tau) \leftarrow 0$ for each location job $\tau \in \mathcal{J}$, $\mathcal{J}' \leftarrow \mathcal{J}$
 2  $x_c \leftarrow \max\{low[I], \min_{\tau \in \mathcal{J}} r(\tau)\}$    ▷ Current location
 3  while $x_c < high[I]$ and $\mathcal{J}' \neq \emptyset$
 4        do $\tau_c \leftarrow \arg\min_{\tau \in \mathcal{J}', r(\tau) \leq x_c} d(\tau)$
 5           $x_1 \leftarrow \min\{\min_{\tau \in \mathcal{J}' \setminus \tau_c} r(\tau), high[I]\}$    ▷ Location where next job released
 6           $v_c \leftarrow \sqrt{v_0^2 + 2a(x_c - low[I])}$    ▷ Speed at $x_c$
 7           $x_2 \leftarrow a_{max} \cdot e(\tau)^2/2 + v_c \cdot e(\tau) + x_c$    ▷ Location where $\tau_c$ completes
 8           if $x_1 < x_2$
 9             then $v_1 \leftarrow \sqrt{v_c^2 + 2a(x_1 - x_c)}$
10                  $e(\tau_c) \leftarrow e(\tau_c) + (v_1 - v_c)/a$
11                  $x_c \leftarrow x_1$
12             else $v_2 \leftarrow \sqrt{v_c^2 + 2a(x_2 - x_c)}$
13                  $a(\tau_c) \leftarrow a(\tau_c) + (v_2 - v_c)/a$
14                  $\mathcal{J}' \leftarrow \mathcal{J}' \setminus \tau_c$
15                  if $\mathcal{J}' \neq \emptyset$
16                     then $x_c \leftarrow \min_{\tau \in \mathcal{J}'} r(\tau)$
17                     else $x_c \leftarrow high[I]$    ▷ End
18  return $a$    ▷ List of allocated time
```

Similarly, the procedure LRT-DECEL applies LRT algorithm for the set $\mathcal{J}$ of location jobs within an interval I, assuming the data mule moves at the speed v_0 at the start of the interval and decelerates at a (i.e., accelerates at $-a$) toward the end of the interval. Internally it is implemented using EDF-ACCEL by reversing the location axis.

```
LRT-DECEL($\mathcal{J}, I, v_0, a$)
1  ▷ Init: $\mathcal{J}_{rev} \leftarrow \emptyset$
2  for each location job $\tau \in \mathcal{J}$        ▷ Make "reversed" set of jobs
3          do $\tau' \leftarrow \{[-d(\tau), -r(\tau)], e(\tau)\}$
4             $\mathcal{J}_{rev} \leftarrow \mathcal{J}_{rev} \cup \tau'$
5  return EDF-ACCEL($\mathcal{J}_{rev}, [-high[I], -low[I]], v_0, a$)
```

Given a set $\mathcal{J}$ of location jobs, an interval I_l, the tight interval I_t, and the plateau speed v_p, the procedure DIVIDE-JOBS returns two sets $\mathcal{J}_l$ and $\mathcal{J}_r$ of location jobs. This procedure is called only when the tight interval does not overlap with accel/decel intervals (Case 2 in Figure 7.6) and thus there are two free intervals before and after the tight interval (denoted I_l and I_r, respectively: only I_l is given as an input). $\mathcal{J}_l$ and $\mathcal{J}_r$ only contain jobs whose feasible location interval is contained in I_l and I_r, respectively.

Internally, it calls either DIVIDE-JOBS-ZERO or DIVIDE-JOBS-NONZERO, depending on the plateau speed. DIVIDE-JOBS-ZERO is called when the plateau speed is zero and thus the tight interval degenerates to a point (denoted x_t). Jobs in $\mathcal{J}$ are divided into $\mathcal{J}_l$ and $\mathcal{J}_r$ at the point x_t. Specifically, the jobs before x_t are added to $\mathcal{J}_l$ and the ones after x_t are added to $\mathcal{J}_r$. Jobs that have a zero-length feasible interval on x_t are not added to either $\mathcal{J}_l$ or $\mathcal{J}_r$. When a job's feasible interval contains x_t, it is divided into two jobs at x_t and each added to $\mathcal{J}_l$ and $\mathcal{J}_r$. The execution time is divided to these two jobs proportionally to the length of their feasible location intervals.

When the plateau speed is greater than zero, DIVIDE-JOBS-NONZERO is called. In this procedure, jobs in $\mathcal{J}$ are divided into $\mathcal{J}_l$ and $\mathcal{J}_r$ by simulating the scheduling by EDF algorithm within I_l. When a job is finished in I_l (i.e., allocated time is equal to the execution time), it is added to $\mathcal{J}_l$. When a job is not scheduled in I_l, it is added to $\mathcal{J}_r$. Otherwise, when a job is scheduled in I_l but not finished yet, it is divided into two jobs, each of which is added to $\mathcal{J}_l$ and $\mathcal{J}_r$, respectively. In that case, the execution time of these two jobs are determined according to the time duration allocated to the job in I_l.

```
DIVIDE-JOBS($\mathcal{J}, I_l, I_t, v_p$)
1  if $v_p = 0$
2     then $x_t \leftarrow low[I_t]$ $(= high[I_t])$    ▷ Tight interval degenerates to a point
3          return DIVIDE-JOBS-ZERO($\mathcal{J}, x_t$)
4     else  return DIVIDE-JOBS-NONZERO($\mathcal{J}, I_l, I_t, v_p$)
```

```
DIVIDE-JOBS-ZERO($\mathcal{J}, x_t$)
 1  ▷ Init: $\mathcal{J}_l \leftarrow \emptyset$, $\mathcal{J}_r \leftarrow \emptyset$
 2  for each location job $\tau \in \mathcal{J}$
 3          do if $r(\tau) < x_t$ and $d(\tau) \le x_t$
 4                then $\mathcal{J}_l \leftarrow \mathcal{J}_l \cup \tau$
 5             elseif $x_t \le r(\tau)$ and $x_t < d(\tau)$
 6                then $\mathcal{J}_r \leftarrow \mathcal{J}_r \cup \tau$
 7             elseif $r(\tau) < x_t$ and $x_t < d(\tau)$
 8                then $I_1 \leftarrow [r(\tau), x_t]$, $I_2 \leftarrow [x_t, high(\tau)]$
 9                     $e_1 \leftarrow \frac{|I_1|}{|I(\tau)|}e(\tau)$, $e_2 \leftarrow \frac{|I_2|}{|I(\tau)|}e(\tau)$        ▷ Proportionally distribute
10                     $\mathcal{J}_l \leftarrow \mathcal{J}_l \cup \{I_1, e_1\}$, $\mathcal{J}_r \leftarrow \mathcal{J}_r \cup \{I_2, e_2\}$
11             else     ▷ Do nothing for zero-length jobs at $x_t$
12  return $\{\mathcal{J}_l, \mathcal{J}_r\}$
```

```
DIVIDE-JOBS-NONZERO($\mathcal{J}, I_l, I_t, v_p$)
1  ▷ Init: $\mathcal{J}_l \leftarrow \emptyset,\ \mathcal{J}_r \leftarrow \emptyset$
2  $a \leftarrow$ EDF-ACCEL($\mathcal{J}, I_l, v_p, 0$)        ▷ Simulate EDF for $I_l$ (acceleration=0)
3  for each location job $\tau \in \mathcal{J}$
4        do if $a(\tau) = e(\tau)$                ▷ Finished in $I_l$
5              then $\tau' \leftarrow \{[r(\tau), \min\{d(\tau), low[I_t]\}], e(\tau)\}$
6                   $\mathcal{J}_l \leftarrow \mathcal{J}_l \cup \tau'$
7           elseif $a(\tau) = 0$                 ▷ Not started in $I_l$
8              then $\tau' \leftarrow \{[\max\{r(\tau), high[I_t]\}, d(\tau)], e(\tau)\}$
9                   $\mathcal{J}_r \leftarrow \mathcal{J}_r \cup \tau$
10          else                                ▷ Dividing into two jobs
11                  $\tau_l \leftarrow \{[r(\tau), low[I_t]], a(\tau)\}$
12                  $\tau_r \leftarrow \{[high[I_t], d(\tau)], e(\tau) - a(\tau)\}$
13                  $\mathcal{J}_l \leftarrow \mathcal{J}_l \cup \tau_l,\ \mathcal{J}_r \leftarrow \mathcal{J}_r \cup \tau_r$
14 return $\{\mathcal{J}_l, \mathcal{J}_r\}$
```

Given a set $\mathcal{J}$ of location jobs and a tight interval I_t, the procedure STOP-TIME returns the time duration the data mule stops at the tight interval I_t. It is called only when I_t degenerates to a single point (and the plateau speed v_p is zero), where some of the jobs has a zero-length feasible location interval. To execute these jobs, the data mule needs to stay at the point for the time equal to the total execution time of these jobs.

```
STOP-TIME($\mathcal{J}, I_t$)
1  ▷ Init: $t_s \leftarrow 0$
2  for each location job $\tau \in \mathcal{J}$
3        do if $I(\tau) \in I_t$
4              then $t_s \leftarrow t_s + e(\tau)$
5  return $t_s$
```

7.2.3 When endpoint speed is unconstrained

Although our heuristic algorithm assumes the speed at both X_s and X_d 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 $[X'_s, X'_d]$ so that we can freely change the speed at X_s and X_d.

1. Simplify the set of location jobs by SIMPLIFY-JOBSET

2. Identify one tight location interval I_t by using the algorithm for CONSTANT SPEED 1-D DATA MULE SCHEDULING (in Section 4.1.2); let g_t denote the processor demand for I_t.

3. Calculate maximum possible speed v_e at the edge of I_t. We can achieve the maximum speed when the data mule constantly increases speed within I_t. Therefore we have the relation $v_e^2 - v_e'^2 = 2a_{max}|I_t|$ and $v_e = v'_e + a_{max}g_t$, and obtain

$$v_e = \frac{a_{max}g_t}{2} + \frac{|I_t|}{g_t} \tag{7.1}$$

4. Calculate maximum possible speed at X_s and X_d. Let v_s and v_d denote the maximum speed for X_s and X_d, respectively. Assuming there is no other tight interval except I_t, v_s is achieved when the speed at $low[I_t](\equiv x_l)$ is v_e. Similarly v_d is achieved when the speed at $high[I_t](\equiv x_h)$ is v_e. Then we have the relations $v_s^2 - v_e^2 = 2a_{max}x_l$ and $v_d^2 - v_e^2 = 2a_{max}(X_d - x_h)$. We solve them and obtain

$$v_s = \sqrt{v_e^2 + 2a_{max}x_l} \tag{7.2}$$
$$v_d = \sqrt{v_e^2 + 2a_{max}(X_d - x_h)} \tag{7.3}$$

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

$$X'_s = X_s - \frac{v_s^2}{2a_{max}} \tag{7.4}$$

$$X'_d = X_d - \frac{v_d^2}{2a_{max}} \tag{7.5}$$

For the hypothetical travel interval $[X'_s, X'_d]$, we run Maximize-Plateau-Speed and recursively maximize the speed, as in the constrained case. Then we trim off the speed changes outside of $[X_s, X_d]$ from the output. Note all of Equations (7.1) to (7.5) are closed-form solutions, and so we can calculate X'_s and X'_d efficiently once we obtain I_t and g_t.

7.3 Analysis

7.3.1 Correctness proof

Claim 1. *For any valid input,* Approx-MotionPlan *halts.*

Proof idea. For every recursion, a tight interval contains at least one job. Thus, in Divide-Jobs, the size of (i.e., number of jobs in) $\mathcal{J}_l$ and $\mathcal{J}_r$ are strictly less than that in $\mathcal{J}$. When the size of $\mathcal{J}$ is zero, Recursive-Maximize returns without recursive calls. Therefore, Approx-MotionPlan halts.

Claim 2. *For any valid input,* Approx-MotionPlan *produces a valid output, i.e., for the motion plan the algorithm produces, the corresponding PREEMPTIVE SCHEDULING FOR GENERAL JOBS problem is feasible.*

Proof idea. Since every problem of GENERALIZED 1-D DATA MULE SCHEDULING is feasible, we show every procedure in the algorithm preserves the feasibility condition.

One iteration of Maximize-Plateau-Speed maximizes the plateau speed until there is a tight interval. Since no interval of $[r(\tau), d(\tau')]$ becomes infeasible in this procedure, if the initial motion plan represents a feasible set of location jobs, the output motion plan is also feasible.

The validity of Trim-Accel-Interval, Trim-Decel-Interval, and Divide-Jobs is due to the optimality of EDF and LRT algorithms. For any given motion plan that is feasible, the corresponding time-domain scheduling problem is schedulable by EDF. Trim-Accel-Interval emulates EDF algorithm from the start to a certain location and trim the feasible intervals of jobs according to the result. Since the remaining set of location jobs is also schedulable by EDF and thus feasible, the procedure preserves the feasibility. Trim-Decel-Interval do the same thing backwards from the end and emulate LRT algorithm, which also works backwards. After running EDF 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. Divide-Jobs-NonZero applies EDF to the plateau interval left of the tight interval. As we discussed, the remaining set of location jobs on the right of the tight interval continues to be feasible. Finally, for Divide-Jobs-Zero, we can divide the feasibility intervals of the jobs arbitrarily without violating feasibility, since the plateau speed is still zero.

7.3.2 Computational complexity

To analyze the computational complexity of the algorithm, we look into the recursive portions of the algorithm. In each recursive step, the most time-consuming operations take $O(m^2)$ time, where m is the number of jobs in $\mathcal{J}$. In Maximize-Plateau-Speed, finding a tight interval (lines 1-2) takes $O(m^3)$ time in a naive implementation, since we need to calculate the processor demand for $O(m^2)$ different location intervals. However, we can do the computation incrementally for each starting location and reduce the time to $O(m^2)$. 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 $O(m)$ time for each starting location, and since there are $O(m)$ starting locations, the whole process of finding a tight interval takes $O(m^2)$.

Note we convert the original set of location jobs to a set of simple location jobs in SIMPLIFY-JOBSET, so m is at most nk, where n is the number of location jobs and k is the maximum number of feasible location intervals of a location job.

We analyze the worst case complexity. For that purpose, we can focus on the case having two recursive calls in RECURSIVE-MAXIMIZE. Since there is at least one location job contained in the tight interval, the following recurrence holds:

$$T(m) = m^2 + \max_{0 \le s \le m-1} (T(s) + T(m-s-1))$$

We show $T(m) = O(m^3)$ by showing there is a constant c, c_0 such that $T(m) \le cm^3$ for $c > c_0$.

$$\begin{aligned}
T(s) + T(m-s-1) &\le c(s^3 + (-s + (m-1))^3) \\
&= c(m-1)\underbrace{(3s^2 + 3s(m-1) + (m-1)^2)}_{\text{max at } s=0 \text{ or } s=m-1} \\
&\le c(m-1)^3 \\
T(m) &\le m^2 + c(m-1)^3 \\
&= cm^3 + (-3c+1)\left(m + \frac{3c}{2(-3c+1)}\right)^2 - \frac{c(3c-4)}{4(3c-1)} \\
&\le cm^3 - \frac{c(3c-4)}{4(3c-1)} \quad (\text{for } c > \tfrac{1}{3}) \\
&\le cm^3 (\text{for } c > \tfrac{4}{3})
\end{aligned}$$

7.3.3 Approximation ratio

We show the approximation ratio for $k \ge 2$ case is not bounded by a constant factor. Figure 7.7 shows an example. Let ε be a small positive value satisfying $2\varepsilon \ll \frac{L}{n}$. Assume the length of each task is $\frac{E}{n}$, where E is a constant. Let T_{opt}, T_{approx} denote the time to finish all the tasks and reach the destination in case of optimal algorithm and the approximation algorithm, respectively.

$$\begin{aligned}
\frac{T_{approx}}{T_{opt}} &\to \frac{n \cdot 2\sqrt{\frac{L/n}{a_{max}}} + E}{2\sqrt{\frac{L}{a_{max}}} + E} \quad (\varepsilon \to 0) \\
&= \frac{2\sqrt{n}\sqrt{\frac{L}{a_{max}}} + E}{2\sqrt{\frac{L}{a_{max}}} + E} \\
&\to +\infty \quad (n \to +\infty)
\end{aligned}$$

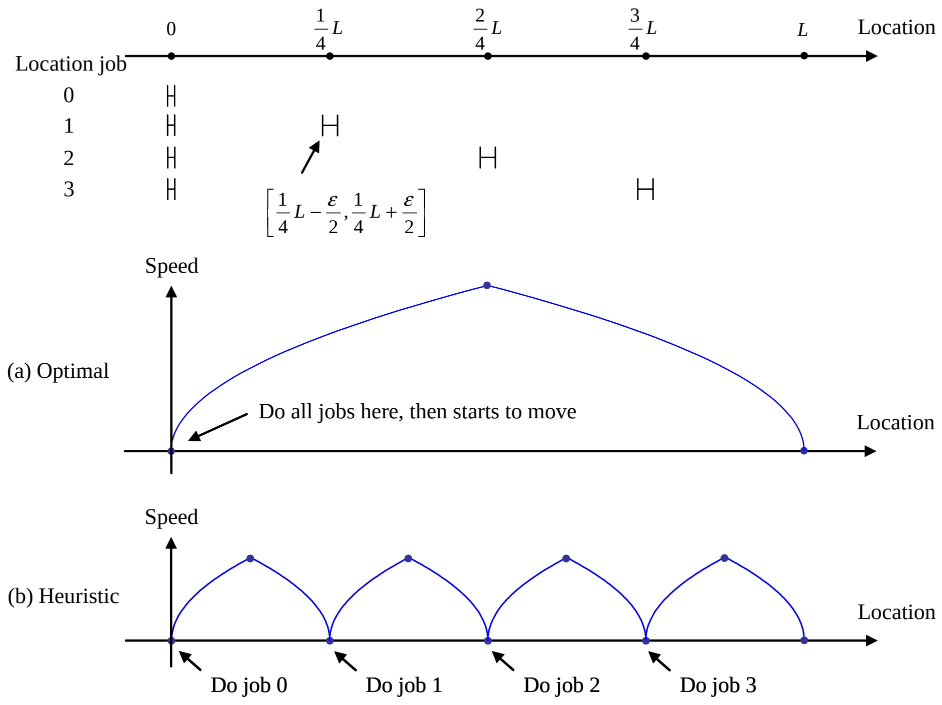


Figure 7.7: An example to show the approximation ratio is unbounded when $k \geq 2$: Case of $n = 4$ location jobs. All feasible location intervals at $x = 0$ are zero-length. All other feasible location intervals have the length of ε. (a) The optimal (i.e., fastest) travel is to first execute all the jobs at $x = 0$ and then start to move. (b) The solution the heuristic algorithm yields. For job 1 to $(n-1)$, the algorithm assigns all the execution time to the latter feasible location intervals, and thus the data mule needs to stop at each $x = \frac{i}{n}L$.

Chapter 8

Experiments

In this chapter, we evaluate the heuristic algorithm for the general case of data mule scheduling problem by numerical experiments.

8.1 Method

We implemented the quadratic program formulation, its relaxation, and the heuristic algorithm in MATLAB with YALMIP interface [Löf04]. We developed and ran the programs on MATLAB 7.4.0 (R2007a) and used SeDuMi [Stu99] version 1.1 for SDP solver.

8.1.1 Test case generation

We randomly generate test cases in the following way. In Figure 8.1, the horizontal dotted line in the middle of the rectangle represents the path of data mule. L is the total travel length. We put circles of diameter d so that each circle has its center inside the rectangle. A circle represents the communication range of a sensor node and the center is its location. 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 d. We put nk circles, where n is the number of location jobs and k is the number of feasible location intervals per each location job. As a result, each location job has at most k feasible location intervals. The number can be less than k, since location intervals from multiple circles may overlap and become one long location interval.

For the experiments, we use $d = 5$ and vary n, k, and L. 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 location job to 10.

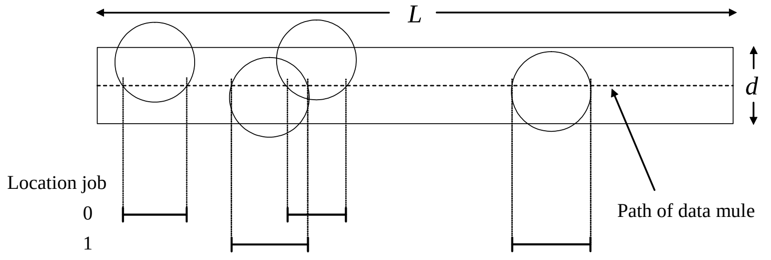


Figure 8.1: Generating test cases: two location jobs with two feasible location intervals per each

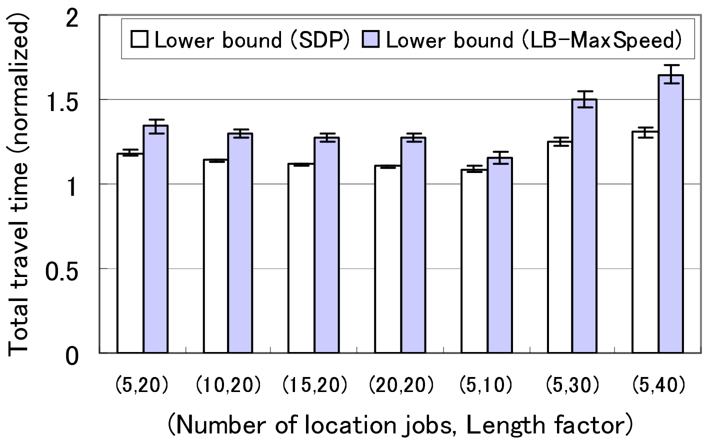


Figure 8.2: Comparison of lower bounds from SDP relaxation and MaxSpeed

8.1.2 Evaluation metrics

We use the following two metrics to evaluate the results:

- **Quality**: total travel time
- **Scalability**: elapsed time to compute the schedule

Since global optimal solutions are hard to obtain for nonconvex quadratic optimization problem like this, we evaluate the heuristic algorithm by comparing it with the lower bounds. Hence, a lower bound should be as tight as possible for the evaluation to be more accurate. In Figure 8.2, we compare the lower bound from SDP relaxation (Section 6.3.1) with the one from LB-MaxSpeed (Section 6.3.2). 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, as LB-MaxSpeed is only for simple location jobs.

We repeat each experiment for 100 times on each case and calculate average and standard deviation.

8.2 Results and discussion

Figures 8.3 and 8.4 are the results for the same problem from the quadratic program and the heuristic algorithm. The parameter of the problem are: number of location jobs $n = 5$, length factor $f = 20$ (i.e. total travel length $L = 20n$), and number of feasible location intervals per job $k = 2$. Note that the result from quadratic program is not necessarily the global optimal solution, as there is no analytical way to test the global optimality in general for nonconvex optimization problems. Also note that we cannot have figures like these for SDP relaxation and LB-MaxSpeed, since they only give lower bounds and do not produce feasible schedules.

8.2.1 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 8.5 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, the schedules from the heuristic algorithm have around 1.15 times longer total travel time than those from LB-MaxSpeed. The ratio slightly increases (1.13 for $n = 5$,

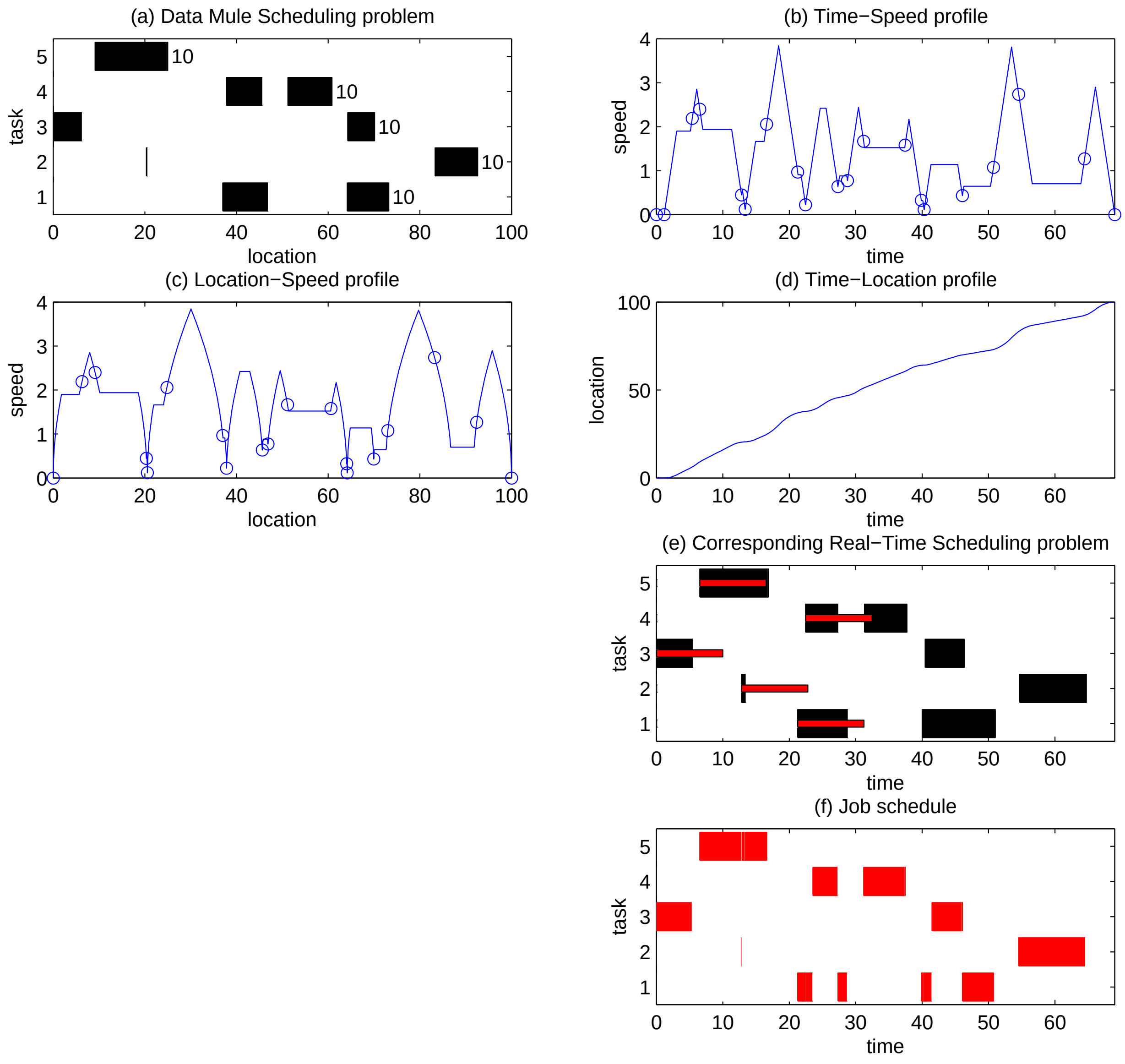


Figure 8.3: Optimal solution (5 jobs, 2 feasible location intervals): total travel time is 68.2948sec. (a) original data mule scheduling problem. number represents the execution time of each location job; (b) a time-speed profile for (a); (c) location-speed profile generated from (b); (d) time-location profile generated from (b); (e) real-time scheduling problem generated from (a) and (d), each thin bar represents the execution time; (f) job schedule for (e)

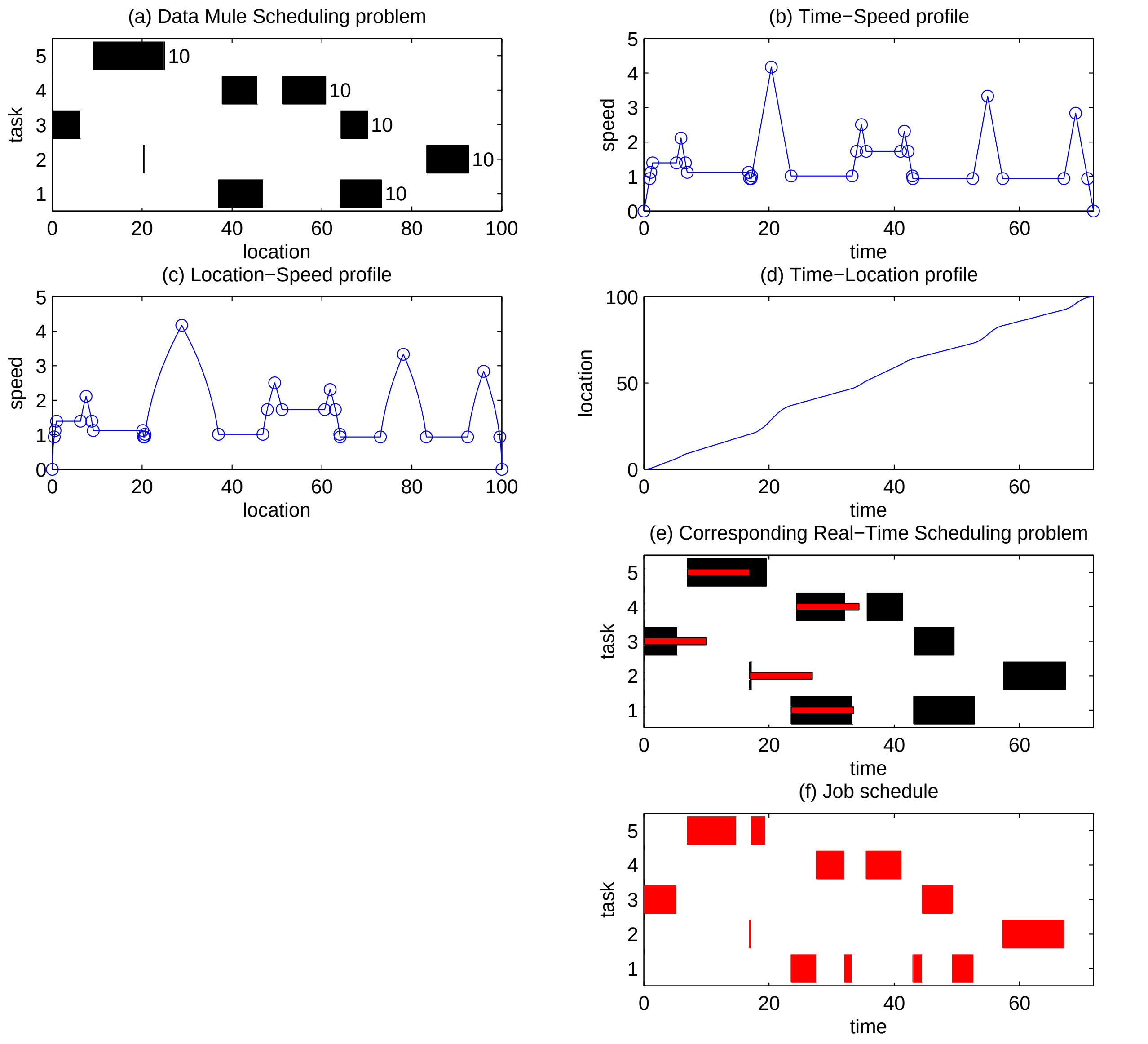


Figure 8.4: Heuristic solution (5 jobs, 2 feasible location intervals): total travel time is 71.8353sec. The legend is same as in Fig 8.3

1.15 for $n = 20$), but it does not imply the heuristic algorithm performs poorer for larger n, since we don't know how close the lower bound is to the optimal solution.

Figure 8.6 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 ratio to the lower bound varies from 1.09 ($f = 10$) to 1.17 ($f = 40$), but we cannot conclude the heuristic algorithm performs poorer for less node density for the same reason as above.

Figure 8.7 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 SDP relaxation. When the number of feasible location interval k is 1, the ratio to the lower bound from LB-MaxSpeed is 1.13 and that for the SDP relaxation is 1.29. As k increases, the ratio decreases (1.23 for $k = 2$, 1.16 for $k = 3$, both to the SDP relaxation).

All these results suggest that the heuristic algorithm produces reasonably good schedules for all the test cases, though we cannot make a strong assertion about the performance from the numerical experiments without having the optimal solution.

8.2.2 Scalability

Figure 8.8 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 0.7 second for 100 location jobs and 2.5 seconds for 200 location jobs. It is reasonably fast, considering the SDP relaxation takes more than 20 minutes on average to compute the lower bound for $n = 20$ case (data not shown). Moreover, the curve suggests the average case time complexity is much better than $O(m^3)$, which is the worst case complexity.

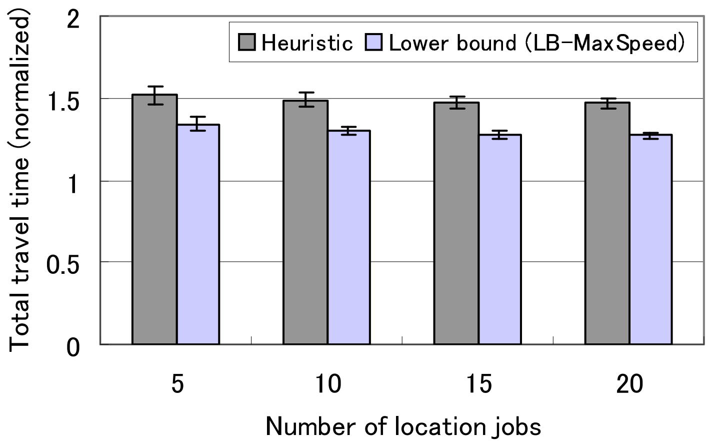


Figure 8.5: Effect of number of location jobs

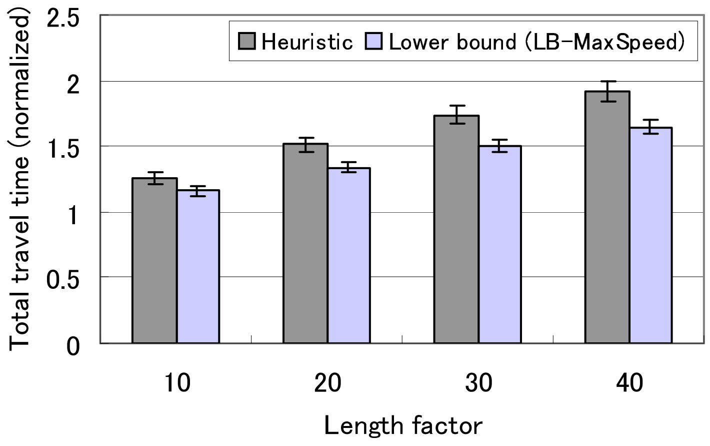


Figure 8.6: Effect of density: node density is less for larger length factor

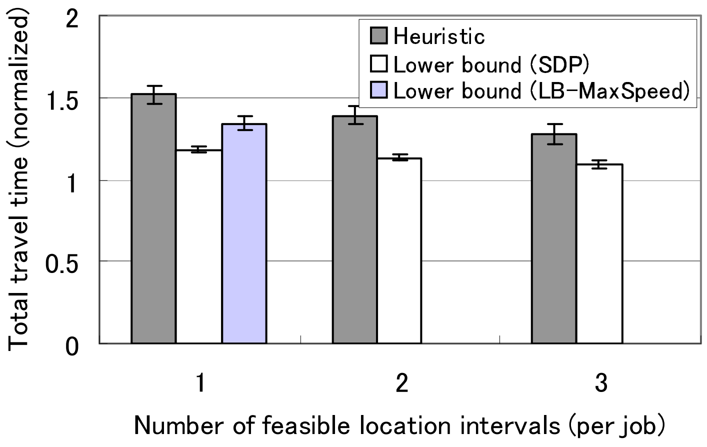


Figure 8.7: Effect of number of feasible location intervals

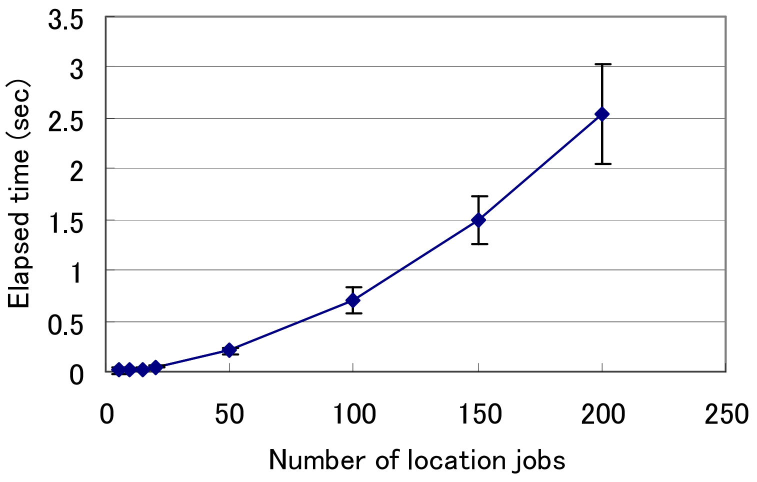


Figure 8.8: Computation time for varying number of location jobs

Chapter 9

Conclusion and Future Work

Unlike traditional multi-hop forwarding, exploiting mobility in collecting data from sensor networks is an alternative way to achieve high energy-efficiency. A mobile node, or "data mule", travels across the sensor field, collects data from each sensor when they are close to each other, and thereby helps reducing energy consumption at each sensor node by avoiding communication over long distance that possible contains multiple hops.

In this paper, we have formulated the problem of planning an optimal movement of data mule as a scheduling problem. This data mule scheduling problem is unique, since it has both time constraints and location constraints characterized by wireless communication range of each sensor node. After reviewing related real-time scheduling problem, we analyzed two simple cases and one general case of data mule scheduling problem. For two simple cases, namely constant speed and variable speed, we presented for each subproblem either an efficient optimal algorithm or a proof of non-existence of an optimal algorithm. Further we pointed out their similarities with speed scaling problems such as DVS. For general case that includes two simple cases as special cases, we proved NP-completeness for general location jobs. We formulated it as a nonconvex quadratic program and then its SDP relaxation to obtain a lower bound. We designed a heuristic algorithm for the general case and showed by numerical experiments that it yields good solutions compared to the lower bound in a reasonable amount of time.

Table 9.1: Summary of complexity results

		Simple (location) jobs		General (location) jobs	
		Offline	Online	Offline	Online
Real-time preemptive scheduling		EDF [LL73]		LP (§3.3.2)	Non-existent (Theorem 3.5)
Data mule scheduling (DMS)	Constant speed (§4.1)	$O(n^2)$	Non-existent	LP (§4.1.2)	Non-existent
	Variable speed (§4.2)	$O(n^3)$ (§4.2.2)	($v_{min} = 0$) EDF-WITH-STOP	LP (§4.2.2)	Non-existent (Theorem 4.3)
			($v_{min} > 0$) Non-existent (Theorem 4.2)		
	Generalized (§5)	Open ($\in \mathcal{NP}$) (Lemma 5.2)	Non-existent	NPC (fixed $k \geq 2$) (Corollary 5.4)	Non-existent
				Strong NPC (k arbitrary) (Corollary 5.6)	

Our study on the data mule scheduling problem is still in its infancy and there remains a huge amount of future work. First of all, there are some open problems. One is the complexity for GENERALIZED 1-D DATA MULE SCHEDULING for simple location jobs. We showed it is in NP but do not know whether it is NP-complete or not. Another open problem is the approximation ratio for the heuristic algorithm. In addition we have left the whole problem of path selection out of focus of this paper. Path selection can be a difficult problem by itself and is even more difficult when it is not clearly separable from other subproblems: for example, when a constraint on turning radius depends on the speed at the time.

Other than these theoretical problems, there are a number of practical issues to resolve when we apply these results in the real-world environment. One is nondeterminism, which we have completely omitted in this paper. Wireless communication can be transient and has fluctuating bandwidth, implying any of the parameters of each location job are in fact nondeterministic. Although we can still see our results as the worst-case analysis, it would be much better if we can design an online heuristic that exploits the uncertainty for improving the quality of the scheduling. Another real-world issue is about overheads. We assumed preemptive scheduling, but it is likely that there is some overhead to initiate the communication for each preemption. There may be additional overhead and other implementation issues when we try to implement the algorithm using conventional communication protocols like ZigBee.

More fundamentally, we can consider a different model for the problem of controlling data mule. For example, we can think of a hybrid of multi-hop approach and data mule approach, as discussed in [MY06]. Combination of these two approaches would reduce the total travel length and the number of jobs, yet each job has longer execution time and energy consumption at each sensor may be higher. We need a different way of formulating the problem, employing another definition of optimality. On higher level of the system, we also need a mechanism for coordinating such communication patterns.

Bibliography

[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.

[BV04] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, New York, NY, USA, 2004.

[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: 11th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 11–16, 2005.

[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.

[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.

[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.

[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.

[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.

[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.

[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.

[SLC03] Chi-sheng Shih, Jane W.S. Liu, and Infan Kuok Cheong. Scheduling jobs with multiple feasible intervals. In *Real-Time and Embedded Computing Systems and Applications (LNCS 2968)*, pages 53–71, 2003.

[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.

[SS84] Barbara Simons and Michael Sipser. On scheduling unit-length jobs with multiple release time/deadline intervals. *Operations Research*, 32(1):80–88, 1984.

[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.

[TAZ06] Muhammad Mukarram Bin Tariq, Mostafa Ammar, and Ellen Zegura. Message ferry route design for sparse ad hoc networks with mobile nodes. In *MobiHoc '06: Proceedings of the 7th ACM international symposium on Mobile ad hoc networking and computing*, pages 37–48, 2006.

[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.

[VB00] Amin Vahdat and David Becker. Epidemic routing for partially-connected ad hoc networks. *Duke University Technical Report*, CS-2000-06, 2000.

[VKR$^+$05] I. Vasilescu, K. Kotay, D. Rus, M. Dunbabin, and P. 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.

[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.

[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.

[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.