The Dynamic Map Visitation Problem: Foremost Waypoint Coverage of Time-Varying Graphs

by

Elliot Keeler Meyerson

Class of 2014

A thesis submitted to the faculty of Wesleyan University in partial fulfillment of the requirements for the Degree of Bachelor of Arts with Departmental Honors in Computer Science

Middletown, Connecticut April, 2014
Abstract

This thesis introduces the Dynamic Map Visitation Problem (DMVP), in which a team of agents must visit a collection of critical locations as quickly as possible, in an environment that may change rapidly and unpredictably during the agents’ navigation. We apply recent formulations of time-varying graphs (TVGs) to DMVP, shedding new light on the computational hierarchy $\mathcal{R} \supset \mathcal{B} \supset \mathcal{P}$ of TVG classes by analyzing them in the context of graph navigation. We provide hardness results for all three classes, and for several restricted topologies, we show a separation between the classes by showing severe inapproximability in $\mathcal{R}$ (recurrent edges), limited approximability in $\mathcal{B}$ (time-bounded recurrent edges), and tractability in $\mathcal{P}$ (periodic edges). We also give topologies in which DMVP in $\mathcal{R}$ is fixed parameter tractable, which may serve as a first step toward fully characterizing the features that make DMVP difficult.
Acknowledgments

I would like to thank the Wesleyan Department of Mathematics & Computer Science for providing me with the intimate community that fostered my interest in these fields. In particular, I’d like to thank Professor Eric Aaron, for setting the vision for DMVP, but even more so for reaching out to me from the start, pushing me to embrace my burgeoning excitement for computer science, and always challenging me to become a better researcher and teacher. I’d like to thank my advisor Professor Danny Krizanc for his constant collaboration and insight on the work in this thesis; for enabling me to be a productive theoretician; and for showing me how to ask interesting and meaningful questions. Thank you to my advisor Professor Wai Kiu Chan for inspiring me to pursue formal science, and for the unfailing support throughout my years at Wesleyan.

Thank you to my family and friends, for guiding me to follow my instincts, and for encouraging me in all my obsessions.

Finally, I’d like to thank the reader, for taking the time to explore my work.
## Contents

Chapter 1. Introduction 1
   1.1. TVG Concepts 4
   1.2. Classes of TVGs 5
   1.3. Classes of Underlying Graphs 6
   1.4. Our Problem 8
   1.5. Our Results 11

Chapter 2. Related Work 13
   2.1. Map Visitation 13
   2.2. Time-varying Graphs 15

Chapter 3. Lower Bounds 18
   3.1. \( \mathcal{R} \): Recurrent edges 18
   3.2. \( \mathcal{B} \): Time-bounded recurrent edges 24
   3.3. \( \mathcal{P} \): Periodic edges 30

Chapter 4. Upper Bounds 34
   4.1. \( \mathcal{R} \): Recurrent edges 34
   4.2. \( \mathcal{B} \): Time-bounded recurrent edges 45
   4.3. \( \mathcal{P} \): Periodic edges 46

Chapter 5. Open Problems and Discussion 54

Bibliography 57
CHAPTER 1

Introduction

Given a set of identified critical locations, how can we coordinate a team of agents to cover these locations as quickly as possible, when the environment is changing rapidly and unpredictably? This question arises across domains as diverse as autonomous robot navigation, wireless sensor network security and maintenance, and AI for non-player characters (NPCs) in video games and virtual worlds. High level approaches to traversal problems in these domains often use waypoint graphs for environment representation, but static graph structures cannot fully capture the heavy environmental dynamics that can occur. This thesis introduces, and presents the first complexity results for, the Dynamic Map Navigation Problem (DMVP), applying recent formulations of highly dynamic graphs (or time-varying graphs (TVGs)) to an essential graph navigation problem: In DMVP, a team of agents must inspect a collection of critical locations on a map (represented by a graph) as quickly as possible, when the environment may change during navigation.

Searching for victims during a hurricane, a team of robots [34, 38, 45] may have a pre-loaded map of the area they are searching, but throughways may become blocked with debris, or become reavailable as debris is cleared by the storm. This suggests the need for a dynamic model of the urban topology. We have a similar situation in urban armed conflict environments. Robots need to search for contraband in environments too dangerous for humans, but streets may be at times too dangerous even for these robots to traverse, depending on local spurts and lulls in violence. This idea can also be extended to the case of
riots, when streets may be impassable due to crowd flux. There are autonomous robot applications of a slightly different feel in planetary exploration [49], in which robots must survey a set of locations for traces of water or other minerals, but atmospheric activity may temporarily limit mobility in certain regions [9]. In the wireless sensor network domain [18, 22, 25], there are several natural applications in which mobile agents must quickly visit a set of critical nodes, for example, for periodic network maintenance, or to inspect for security breaches and vulnerabilities. In ad-hoc wireless networks, the topology of the network may be changing during its operation. In a video game or virtual world [23, 24, 26], it may be productive for characters to visit a set of previously identified locations to collect valuable items, such as weapons, money, power-ups, etc. Non-player characters are often given complete game maps, but maps may change, as a result of creative/destructive character/environmental behaviors, and traversability of maps may be hindered as sporadic violence makes some edges temporarily too dangerous to use. In all of the above domains, we come across the same issue: we have an intuitive way to model the environment with a graph, but we need to somehow incorporate into our model potentially high levels of environmental dynamics.

When incorporating dynamics into a problem such as DMVP, there are many options for how to constrain/model the dynamics of the graph. Dynamics can be deterministic (e.g., [13, 32, 33, 42, 51]) or stochastic (e.g., [8, 15]). In this thesis, to provide a foundation for future work, and exemplify the aspects of topologies and dynamics that make our problem (Section 1.4) easy or hard, we focus on the deterministic case. The deterministic approach is also particularly relevant for situations in which some prediction of changes is feasible. Quite a bit of this previous work has required that the graph be connected at all times [15, 33, 40]. Indeed, for complete map visitation to be possible, every critical location must be
eventually reachable. However, in application environments such as those outlined above, at any given time the waypoint graph may be disconnected. Our model must be general enough to allow for this phenomenon.

We adopt three classes of TVGs, each of which places constraints on edge dynamics. In class $\mathcal{R}$, edges must reappear eventually; in class $\mathcal{B}$, edges must appear within some time bound; in class $\mathcal{P}$ edge appearances are periodic. These classes have proven to be critical points in the TVG taxonomy \cite{14}. They have been studied with respect to problems such as broadcast \cite{13} and exploration \cite{28, 32}, with results relating to feasibility of computation and bounds on broadcast and exploration time. $\mathcal{R}$, $\mathcal{B}$, and $\mathcal{P}$ place intuitive constraints on the nature of dynamic navigation domains. Even the assumption of periodicity of edges has applications to navigation in transportation networks \cite{28, 32}, as well as environments periodically patrolled by other agents, who can prohibit or guarantee safe traversal of an edge.

In this thesis, we shed further light on the computational hierarchy of $\mathcal{R}$, $\mathcal{B}$, and $\mathcal{P}$, by analyzing them in the context of DMVP, a natural problem in global navigation. We provide hardness results for all three classes. For several restricted topologies, we demonstrate separation between the classes by showing severe in-approximability in $\mathcal{R}$, limited approximability in $\mathcal{B}$, and tractability in $\mathcal{P}$. We also give topologies in which DMVP in $\mathcal{R}$ is tractable and fixed parameter tractable, which may serve as a first step towards fully characterizing the topological features that make DMVP difficult. Because our goal in this thesis is to cleanly differentiate the classes of dynamics we are exploring, rather than explore the interactions between multiple agents, our results here focus on the case of a single agent, though we give a few extensions to $k$ agents.
1.1. TVG Concepts

We adopt a recent unified model of time-varying graphs [14], which is general enough to capture the dynamics of any of the aforementioned applications.

**Definition 1.** A TVG (time-varying graph, dynamic graph, or dynamic network) is a five-tuple $G = (V, E, T, \rho, \zeta)$, where $T \subseteq \mathbb{T}$ is the *lifetime* of the system, presence function $\rho(e, t) = 1 \iff$ edge $e \in E$ is available at time $t \in T$, and latency function $\zeta(e, t)$ gives the time it takes to cross $e$ if starting at time $t$. There is also the possibility for a similarly defined node presence function $\psi$, and node latency function $\varphi$. The graph $G = (V, E)$ is called the *underlying graph* of $G$, with $|V| = n$.

In the most general case, $T$ can be $\mathbb{R}$, and edges can be directed. However, in our work we consider the discrete case in which $T = \mathbb{N}$, edges are undirected, and all edges have uniform travel cost 1. Furthermore, we consider only the case of dynamic edges, that is, we assume all vertices of $G$ are present for all $t \in T$. This core model is in line with recent work in TVG exploration [15, 28, 32, 33].

**Definition 2.** A *temporal subgraph* of a TVG $G$ results from restricting the lifetime $T$ of $G$ to some $T' \subseteq T$. A *static snapshot* of $G$ is a temporal subgraph with static edges.

**Definition 3.** $J = \{(e_1, t_1), \ldots, (e_k, t_k)\}$ is a journey $\iff \{e_1, \ldots, e_k\}$ is a walk in $G$ (called the *underlying walk* of $J$), $\rho(e_i, t_i) = 1$ and $t_{i+1} \geq t_i + \zeta(e_i, t_i)$ for all $i < k$ (Note: the final criterion is not required for our discrete case). $t_1$ is called the *departure date* of $J$, and $t_k$ is called the *arrival date*. The *topological length* of $J$ is $k$, the number of edges traversed. The *temporal length* is the duration of the journey: $(arrival \; date) - (departure \; date)$. If a journey exists between nodes $u$ and $v$, we say $u$ can reach $v$, and write $u \leadsto v$. 
The following three notions of minimal journeys are fundamental for optimization problems over TVGs \([13, 14, 51]\).

**Definition 4.** Given a date \(t\), a journey from \(u\) to \(v\) departing on or after \(t\) whose arrival date \(t'\) is soonest is called *foremost*; whose topological length is minimal is called *shortest*; and whose temporal length is minimal is called *fastest*.

### 1.2. Classes of TVGs

In \([14]\), a hierarchy of thirteen classes of TVG’s is presented. In related work on exploration \([28]\) and broadcast \([13]\), focus is primarily on computational separations in the chain \(\mathcal{R} \supset \mathcal{B} \supset \mathcal{P}\) defined below. We adopt these classes into our domain, which we believe enforce natural constraints in our application environments.

**Definition 5.** (Recurrent edges) \(\mathcal{R}\) is the class of all TVG’s \(G\) such that \(G\) is connected, and \(\forall e \in E, \forall t \in T, \exists t' > t\) s.t. \(\rho(e, t') = 1\).

**Definition 6.** (Time-bounded recurrent edges) \(\mathcal{B}\) is the class of all TVG’s \(G\) such that \(G\) is connected, and \(\forall e \in E, \forall t \in T, \exists t' \in [t, t + \Delta)\) s.t. \(\rho(e, t') = 1\), for some \(\Delta\).

**Definition 7.** (Periodic edges) \(\mathcal{P}\) is the class of all TVG’s \(G\) such that \(G\) is connected, and \(\forall e \in E, \forall t \in T, \forall k \in \mathbb{N}, \rho(e, t) = \rho(e, t + kp)\) for some \(p\). We call \(p\) the *period* of \(G\).

In other words, in \(\mathcal{R}\), edges must reappear eventually; in \(\mathcal{B}\), edges must appear within some time bound; in \(\mathcal{P}\) edge appearances are periodic. For the cases of \(\mathcal{B}\) and \(\mathcal{P}\), we assume \(\Delta\) and \(p\) are integers.
1.3. Classes of Underlying Graphs

The following terms are used to describe underlying graphs of TVGs in our results. As much as possible, we take standard notation and terms from the graph theory literature, e.g., [12]. In this thesis, all graphs are undirected.

**Definition 8.** A tree is a connected graph with \( n - 1 \) edges.

**Definition 9.** A path is a tree of max degree 2. In particular, the \( n \)-vertex path graph \( P_n \) consists solely of vertices \( v_1, \ldots, v_n \) and edges \( (v_i, v_{i+1}), i \in \{1, \ldots, n-1\} \) (Figure 1).

![Figure 1. \( P_6 \)](image)

**Definition 10.** A cycle is a connected graph in which all vertices have degree 2 (i.e., it is 2-regular). In particular, the \( n \)-vertex cycle \( C_n \) is obtained by adding the edge \( (v_n, v_1) \) to \( P_n \) (Figure 2).

![Figure 2. \( C_6 \)](image)

**Definition 11.** A star is a tree in which at most one vertex has degree greater than one. The \( n \)-leaf star \( S_n \) consists solely of vertices \( c, v_1, \ldots, v_n \) and edges \( (c, v_i), i \in \{1, \ldots, n\} \) (Figure 3).

![Figure 3. \( S_6 \)](image)
**Definition 12.** A *spider* is a tree in which at most one vertex has degree greater than two. In other words, a spider consists of a set of vertex-disjoint paths, called *arms*, each of which has exactly one endpoint connected to the common central vertex $c$ (Figure 4).

![Figure 4. Six-armed spider](image)

**Definition 13.** A *comb* is a max-degree 3 tree, in which there exists a simple path containing every vertex of degree 3 (Figure 5). Such a path is called a *backbone* of the comb. Paths edge disjoint to the backbone are called *arms*. A leaf distance 1 from the backbone is called a *tooth*.

**Definition 14.** An *$r$-almost-tree* is a connected graph with $|V| + r - 1$ edges, that is, $r$ edges can be removes to produce a tree (Figure 6). More generally, a graph with $c$ connected components is said to have *circuit rank* $r = |E| - |V| + c$. 

1.4. OUR PROBLEM

We consider the following problem.

**Problem.** Given a dynamic graph $G$ and a multiset of starting locations $S$ for $k$ agents in $G$, the TVG foremost coverage or dynamic map visitation problem $DMVP(G, S)$ is the task of finding journeys for each of these $k$ agents such that every node in $V$ is in some journey, and the maximum temporal length among all $k$ journeys is minimized. The decision variant $DMVP(G, S, t)$ asks whether this coverage can be completed in no more than a given $t$ time steps, that is, these journeys can be found such that no journey has temporal length greater than $t$.

**Definition 15.** The *edge-induced subgraph* of a set $\mathcal{J}$ of journeys over underlying graph $G$, is the subgraph consisting of exactly the vertices and edges in the journeys in $\mathcal{J}$.

**1.4. Our Problem**

We consider the following problem.

**Problem.** Given a dynamic graph $G$ and a multiset of starting locations $S$ for $k$ agents in $G$, the TVG foremost coverage or dynamic map visitation problem $DMVP(G, S)$ is the task of finding journeys for each of these $k$ agents such that every node in $V$ is in some journey, and the maximum temporal length among all $k$ journeys is minimized. The decision variant $DMVP(G, S, t)$ asks whether this coverage can be completed in no more than a given $t$ time steps, that is, these journeys can be found such that no journey has temporal length greater than $t$. 
For the minimization problem $\text{DMVP}(G, S)$ and the corresponding decision problem $\text{DMVP}(G, S, t)$, input is viewed as a sequence of graphs $G_i$ each represented as an adjacency matrix, with an associated integer duration $t_i$, i.e., $G = (G_1, t_1), (G_2, t_2), \ldots, (G_m, t_m)$, where $G_1$ appears initially at time zero (see [41] and [14] for other views of dynamic graphs). Let $T = \sum_{i=1}^{m} t_i$. Note that since each $t_i$ can be encoded in $O(\log t_i)$ space, it is possible for $T$ to be exponential in the size of $G$. The following observation is required to show that the number of time steps of $G$ that need to be considered for DMVP is in fact polynomial in the size of $G$.

**Observation 1.** When computing DMVP over $G$, it is not necessary to consider each static temporal subgraph $(G_i, t_i)$ for more than $2n - 3$ time steps.

**Proof.** Suppose $G_i$ is the available static subgraph of $G$ from times $\tau$ to $\tau + t_i$, and $t_i > 2n - 3$. Suppose agent $a$ starts at vertex $u$ at time $\tau$. There are two cases:

Case 1: If $a$ can complete its coverage of $G$ by only traversing in $G_i$, then in the worst case $a$ can execute any complete spanning tree traversal of $G_i$, which takes no more than $2n - 3$ steps. In this case, it does not matter at which vertex $a$ ends up, because the task has been completed.

Case 2: If there is a vertex $v$ such that $a$ has not covered $v$ by time $\tau$, and $u$ and $v$ are in different connected components in $G_i$, then $a$ cannot complete coverage of $G$ when $G_i$ is the available static subgraph. In this case it may matter which vertex $a$ ends up at, depending on which future edges will be available. The size of the connected component of $u$ in $G_i$ is at most $n - 1$, so a spanning tree traversal of this component ending up back at $u$ takes no more than $2n - 4$ steps. If $a$ would rather end up at a different vertex $w \neq u$, it simply traverses $w$’s branch of the spanning tree last, and returns up only to $w$, in fewer than $2n - 4$ steps. □
By Observation 1, for any \( t_i > 2n - 3 \), when computing DMVP, all time steps after the first \( 2n - 3 \) can be ignored (skipped). DMVP over \( G \) can be computed by computing DMVP over \( G' = (G_1, \min(t_1, 2n - 3)), \ldots, (G_m, \min(t_m, 2n - 3)) \), and adding back the cumulative time skipped before completion. \( G' \) can clearly be derived from \( G \) in \( O(m) \) time. The total duration of \( G' \) is \( T' = \sum_{i=1}^{m} \min(t_i, 2n - 3) < 2nm - 3m \), which is polynomial in \( |G| \). Let \( \epsilon(\tau) \) be the time skipped through time \( \tau \in T' \). \( \epsilon(\tau) \) can be simply calculated for all \( \tau \in T' \) in \( O(T') \) time. Pseudocode for this computation is given below (Algorithm 1). Note: a similar \( O(T') \) preprocessing step can be run to associate each time \( \tau \in T' \) with the corresponding available static graph \( G_i \), enabling \( O(1) \) edge presence lookups \( \rho(e, \tau) \).

---

**Algorithm 1** SkippedTime(\( G = (G_1, t_1), \ldots, (G_m, t_m) \))

\[
\begin{align*}
\text{skipped} &= 0 \quad \triangleright \text{Number of time steps skipped so far} \\
\text{start} &= 0 \quad \triangleright \text{Start time of current static graph } G_i \\
\text{for } i = 1, \ldots, m \text{ do} & \\
\quad \text{for } j = 1, \ldots, \min(t_i, 2n - 3) \text{ do} & \triangleright \text{This loop is entered exactly } T' \text{ times.} \\
\quad & \quad \epsilon(\text{start} + j) = \text{skipped} \\
\quad \text{if } j > 2n - 3 \text{ then} & \\
\quad & \quad \text{skipped} = \text{skipped} + t_i - (2n - 3) \\
\quad & \quad \text{start} = \text{start} + \min(t_i, 2n - 3) \\
\text{return } \epsilon
\end{align*}
\]

Since none of the algorithms we present run in \( \Omega(T') \) time, we can run this preprocessing step for every instance of DMVP and not affect the asymptotic running time. Therefore, for the sake of simplicity, for the rest of our results we assume that this preprocessing has taken place, i.e., we think of \( G \) as \( G' \) and \( T \) as \( T' \), thereby avoiding the exponential nature of \( T \). Note also that for the case of \( P \), the constraint of periodicity implies that it is only necessary to look at \( p \) consecutive time steps of the input. The general form for a complete DMVP algorithm with preprocessing (DMVP-PP, Algorithm 2) is given below, demonstrating how natural it is to implement this preprocessing in all cases.
1.5. Our Results

An overview of our results is given below. The results in the table are for the case of DMVP for a single agent, though we extend some of these cases to \( k \) agents (Theorems 8, 10, and Corollary 2). When not stated, ‘DMVP’ refers to DMVP for a single agent.

<table>
<thead>
<tr>
<th>Class</th>
<th>Lower Bounds</th>
<th>Upper Bounds</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \mathcal{R} )</td>
<td>NP-hard to approx. within any factor when restricted to star (Thm.1), max degree 3 tree (Thm.2)</td>
<td>path ( O(T) ) (Thm.7) cycle ( O(Tn) ) (Thm.9) general graphs ( O(Tn^3 + n^22^n) ) (Thm.11) ( m )-leaf ( O(1) )-almost-tree ( \in ) FPT (Thm.12) ( O(\log n) )-leaf ( O(1) )-almost-tree ( \in ) P (Cor.1) spider with no waiting ( \in ) P (Obs.2)</td>
</tr>
<tr>
<td>( \mathcal{B} )</td>
<td>NP-hard to approx. within ( \Delta ) ( \forall \Delta &gt; 1 ) when restricted to spider (Thm.3), max degree 3 trees (Thm.4)</td>
<td>( \Delta )-approx. for trees (Thm.13) 2( \Delta )-approx. for general graphs (Thm.14)</td>
</tr>
<tr>
<td>( \mathcal{P} )</td>
<td>( p = 1 ) generalizes ham-path(Thm.5) There exists a nontrivial class of graphs for which ( p = 2 ) is NP-hard, but ( p = 1 ) is not. (Thm.6)</td>
<td>trivial on wide-spaced comb (Thm.15) spider ( \in ) P (Thm.16) tree ( O(n) ) when ( p = 2 ) (Thm.17)</td>
</tr>
</tbody>
</table>

We show that DMVP in \( \mathcal{R} \) is NP-hard to approximate within any factor, when the underlying graph \( G \) is restricted to a star or tree of max degree 3. We show that in \( \mathcal{B} \) this problem is NP-hard to approximate within any factor less than \( \Delta \), when \( G \) is restricted to a spider or tree of max degree 3. We show that in \( \mathcal{P} \),
DMVP is NP-complete when $p = 1$, and that there is a nontrivial class of graphs for which the problem is NP-hard when $p = 2$, but trivial when $p = 1$.

We show that in $R$, DMVP is solvable in $O(T)$ time when $G$ is a path, $O(Tn)$ when $G$ is a cycle, and for $k$ agents, we generalize these to $O(Tn)$ and $O(T\frac{n^2}{k})$, respectively. For general graphs, we show DMVP is solvable in $O(Tn^3 + n^{222})$ time. Furthermore, in $R$, DMVP is fixed parameter tractable when $G$ is an $m$-leaf $O(1)$-almost tree (for fixed parameter $m$ for a single agent, $m$ and $k$ for $k$ agents), and poly-time solvable when $m = O(\lg n)$. In $B$, we demonstrate a tight $\Delta$-approximation for trees, and a $2\Delta$-approximation for general graphs. We demonstrate a class of problems which are NP-hard in $B$, but solvable by an online algorithm in $P$. Finally, we show that DMVP in $P$ is solvable in polynomial time when $G$ is a spider, for fixed $p$, and we show that when $p = 2$, DMVP is solvable in linear time on general trees.
CHAPTER 2

Related Work

2.1. Map Visitation

The map visitation problem (MVP) was first introduced in [1], with initial complexity and approximation results over a variety of graph topologies including paths, cycles, trees, and general graphs. DMVP generalizes MVP to environments modeled by TVGs. Since automatic exploration was first introduced in [47], problems whose solutions entail the complete visitation of some set of points have been studied in a variety of contexts.

The map visitation problem itself is an essential variation of the traveling salesman problem. The following list of previously studied problems of this sort that are closely related to, but distinct from MVP, is presented in [1]:

Minimum Path Cover Problem [6, 7, 31, 43, 44]. Given a graph find the minimum number of vertex-disjoint paths that cover the graph, i.e., each vertex of the graph is on some path. Here the emphasis is on the number of paths (or agents) required not on the lengths of the paths.

The $k$-Core Problem [37, 50]. A $k$-core of a graph is a set of $k$ disjoint paths such that the distance of any vertex in the graph to some vertex on the paths is minimized. Here the number of paths is fixed, the requirement being that every vertex be near at least one path but not necessarily visited by an agent.

The $k$-Chinese Postman Problem and $k$-Rural Postman Problem [11, 20]. In the $k$-Chinese Postman Problem, $k$ agents starting from the same depot must choose $k$ tours (returning to the depot) which together visit all of the edges of a
graph at least once. In the \(k\)-Rural Postman Problem, not all but a designated subset of the edges of the graph must be visited by at least one postman tour. The standard version of the problem asks that the sum of the length of the tours be minimized but the case of minimizing the maximum length of any tour is also considered \cite{17}. The requirements that the agents visit edges rather than nodes and that they all return to the depot distinguishes this problem from MVP.

The \(k\)-Traveling Repairman Problem \cite{27}. In the \(k\)-Traveling Repairman Problem, \(k\) agents starting from the same depot must choose \(k\) tours which visit all of the vertices of a graph at least once. Among all such choices of tours, the one that minimizes the sum of the lengths of the tours is sought. Minimizing the sum of the lengths of the tours rather than the minimizing the maximum length of any tour and requiring the repairmen to return to a common depot distinguishes this problem from MVP.

Coverage has also been studied in the local robot navigation realm, in which the underlying graph is usually embedded in a grid. Grids tend to be useful for decomposing continuous regions to facilitate efficient computation. Efficient algorithms for multi-agent grid coverage, based on spanning tree decompositions, have been studied in \cite{52} and \cite{2}. For area coverage, spanning tree decompositions can be quite useful. For example, if a robot is half the diameter of cells in the grid, it can trace a path around the spanning tree, covering the entire grid, without needing to retrace its steps. However, in our case of waypoint coverage, graphs are not necessarily gridlike, so many of the assumptions inherent to area coverage problems do not hold.

One interesting case in which visitation has been studied over general graphs is that of ant coverage \cite{48,49}. Ants navigating on a graph are aggressively local agents that can at any time only see nodes that are their immediate neighbors. However, ants leave *pheromone traces* (represented by quantities) at nodes,
which are used to indirectly communicate with other ants. To complete coverage, ants perform local gradient descent, traveling to neighboring nodes with fewer pheromones, in search of unvisited nodes. There are several intriguing open questions related to ant coverage algorithms [39]. It would be interesting to study ant-based algorithms over TVGs, but for now we restrict ourselves to a centralized offline approach.

2.2. Time-varying Graphs

We work from a recently formalized framework of time-varying graphs [14] which generalizes and unifies models presented in a slew of related work, in which closely related models were independently defined under different names, such as *time-varying networks* [28], *dynamic networks* [40, 41, 51], *dynamic graphs* [8, 33, 42, 46], and *evolving graphs* [41, 15]. [14] and [41] present recent surveys of computation over both deterministic and stochastic TVGs.

Like our work, [51] focuses on centralized offline deterministic computation over dynamic networks. They present fundamental results for shortest path-type problems, namely, algorithms for computing shortest, fastest and foremost journeys in a centralized setting. They use a distinct data structure for storing dynamic networks, which favors sparse TVGs, and facilitates certain algorithmic techniques. Because of this difference, and the fact that results are not given for all-pairs or all-times, our presentation of a new algorithm for all-pairs-all-times-foremost-journey (Algorithm 5) does not directly draw from their work. However, [51] is critical to the TVG literature, as it has stimulated interest in deterministic computation, both centralized and distributed, over this sort of structure.

Deterministic distributed computation in dynamic networks is addressed in [40], with worst-case results for standard distributed network problems, including
2.2. TIME-VARYING GRAPHS

counting (the size of the network), token dissemination, and election. They introduce \textit{T-interval connectedness} to constrain network dynamics, which strengthens the notion of a network being \textit{always connected}, i.e., connected at every time step. A dynamic network is \textit{T}-connected if for every \textit{T} consecutive rounds there exists a stable connected spanning subgraph. This constraint enables progress and completion guarantees. Extensions to this constraint model are presented in [41]. However, for the applications we have in mind (see Chapter 1), we believe that possible disconnectedness of static snapshots is a crucial environmental occurrence. So, in this thesis, we do not study constraints that assume always connectedness.

Stochastic methods have yielded another set of intuitive models of TVGs. [15] analyzes the cover time of types of random walks on several models of TVGs, both deterministic and stochastic. The stochastic models are \textit{markovian evolving graphs}, in which the static graph at each time step depends only on the directly preceding graph, and \textit{bernoulli evolving graphs}, in which the static graph at each time step is chosen from an identical Erdős-Rényi random graph distribution [21]. In all cases, [15] assumes always connectedness. [8], on the other hand, takes an \textit{edge markovian} approach to stochastic TVGs, while addressing the question of expected flooding time. The resulting model skirts the need for always connectedness. An edge markovian TVG has some \textit{birth rate} \( p \) and \textit{death rate} \( q \). If an edge is not present in the \textit{i}th step, \( p \) is the probability it will be available during the \( i + 1 \)st step; if it is present in the \textit{i}th step, \( q \) is the probability it will not be available during the \( i + 1 \)st step. Although this thesis focuses on deterministic dynamics, we believe edge markovian models along the lines of, but perhaps less uniform than this one could accurately capture real world environments in which deterministic foresight is impossible.
The problem most closely related to ours that has been studied over TVGs is *exploration*, in which agents must visit every node in a previously *unknown* environment. Although exploration of TVGs has only been studied as an online problem, the motivation for studying exploration is also motivation for studying visitation via other approaches. [15] focuses on exploration from the perspective of cover time of random walks on TVGs, with some intriguing results including a method for avoiding exponential cover time on worst-case deterministic TVGs. In their model, graphs are always connected, yet deterministic coverage algorithms cannot guarantee completion, because there is no constraint on individual edge behavior, and thus agents acting deterministically can be restrained to the endpoints of a single edge. [33] considers exploration of the cycle (or *ring*) with the *T*-interval connected constraint model, and considers both cases when graph dynamics are known and when they are unknown. In the unknown case, to guarantee completion, they also assume time-bounded recurrence of edges (see $\mathcal{B}$, Section 1.2). [28] and [32] consider exploration in the periodic case (see $\mathcal{P}$, Section 1.2), with applications to transportation networks. In [28], [32] and [33], worst-case bounds are given for the time it takes for agents to explore the entire network, given varying sets of constraints and agent knowledge.

Although $\mathcal{R}$, $\mathcal{B}$, and $\mathcal{P}$ (Section 1.2) have each been studied independently, the computational hierarchy $\mathcal{R} \supset \mathcal{B} \supset \mathcal{P}$ itself has also been investigated, with respect to deterministic online computation of shortest, fastest and foremost broadcast [13]. Feasibility and worst-case completion time results across $\mathcal{R}$, $\mathcal{B}$, and $\mathcal{P}$ strictly separate these three classes for each type of broadcast. In this thesis, we show that such a fundamental separation exists also in the case of offline computation of a natural, but generally difficult problem.
CHAPTER 3

Lower Bounds

As motivation for many of the results that follow, it is important to note that MVP [1] for a single agent is solvable in linear time on trees.

3.1. \( \mathcal{R} \): Recurrent edges

To characterize the difficulty of DMVP in \( \mathcal{R} \), we first show inapproximability over stars.

**Theorem 1.** DMVP for a single agent in \( \mathcal{R} \) is NP-hard to approximate within any factor, even when the underlying graph is a star.

**Proof.** We reduce from the set cover problem (SCP). Given a universe \( U = \{1, 2, ..., m\} \), a family \( S \) of subsets \( s_1, s_2, ..., s_n \) of \( U \), and an integer \( k \), it is NP-complete to decide whether \( S \) contains a cover of \( U \) of size \( k \) or less [35]. Given an instance of SCP, construct a star \( G = (V, E) \) with central vertex \( c \); points \( v_1, ..., v_m \), corresponding to elements in \( U \); \( p_1, ..., p_n \), corresponding to sets in \( S \); and a single check point \( p_0 \). We use the following static subgraphs to construct a TVG \( G \). For all \( s_i \) in \( S \), let \( pass(i) = (V, \{c, p_i\}) \), and let \( take(i) = (V, E_i) \), where \( E_i = \{(c, v_j) : j \in s_i\} \). Let \( check = (V, \{c, p_0\}) \). Let \( finish = (V, F) \), where \( F = \{(c, p_i) : 1 \leq i \leq n\} \). Consider the TVG \( G = (pass(1), 1), (take(1), t_1), (pass(1), 1), ..., (pass(n), 1), (take(n), t_n), (pass(n), 1), (check, 2), (finish, 2k-1) \), where \( t_i = 2|s_i|, \forall i \in \{1, ..., n\} \). The total duration of \( G \) is \( D = 2n + 2 \sum_{i=1}^{n}|s_i| + 2k + 1 < 2n + 2mn + 2k + 1 \).

Consider the problem of deciding if DMVP over \( G \) with a single agent \( a \) starting at \( c \) has a solution of length no more than \( D \). This problem is in NP, since given
a journey over $G$, we can easily check that it hits every vertex, and that all of its edges are available at the correct times. Such a solution can be no longer than $D$, since $D$ is the total duration of $G$.

Suppose $S$ contains a cover $C$ of $U$ of size $k$ or less. Then for all $s_i \in C$, a takes $s_i$, that is, $a$ waits at $c$ during both instances of pass$(i)$, and visits all $v_j \in s_i$ and returns to $c$ during take$(i)$, which is possible since the duration of take$(i)$ is $2|s_i|$. Since $C$ is a cover of $U$, $a$ visits all $v_i$. For all $s_i \notin C$, a passes $s_i$, that is, $a$ moves from $c$ to $p_i$ during the first pass$(i)$, waits at $p_i$ during take$(i)$, and returns to $c$ during the second pass$(i)$. During check, $a$ moves from $c$ to $p_0$, then back to $c$. At this point, since $|C| \leq k$, $a$ has passed at least $n - k s_i$'s. So, there are no more than $k p_i$'s left unvisited. $a$ visits these during finish, thus completing visitation of all vertices of $G$ in no more than $D$ steps (e.g., Table 1).

Suppose there exists a solution to this instance of DMVP of length no more than $D$. Prior to finish, $a$ must have visited at least $n - k p_i$'s, since finish only lasts for $2k - 1$ steps. So $a$ must have passed at least $n - k s_i$'s. Taking and passing for a single $s_i$ are mutually exclusive, because if $a$ moves to $p_i$ during the first pass$(i)$, $a$ must wait during take$(i)$, and if $a$ both takes $s_i$ and moves to $p_i$ during the second pass$(i)$, $a$ will be trapped at $p_i$ until finish, and will never be able to reach $p_0$, which must be visited during check, the only input time at which $p_0$ is available. Thus, $a$ could have taken no more than $k s_i$'s. During these $k$ or fewer takes, $a$ must have covered all $v_1, \ldots, v_m$. So, the union of these $k$ or fewer take$(i)$'s contains all edges $(c, v_j)$, which implies that the corresponding $s_i$'s form a cover of $U$ or size $k$ or less. Hence, the decision problem is NP-complete.

Consider the minimization version of the problem with the same setup. Since it is NP-hard to decide if there is a solution of length $D$ or less, it NP-hard to find such a solution. But after $D$ steps, $a$ may have to wait arbitrarily long time for the next edge is a feasible solution to appear, so any feasible solution that takes
longer than $D$ steps can be arbitrarily long. Therefore, the problem cannot be approximated within any factor. □
This inapproximability also holds over the restriction of underlying graphs to trees of max-degree 3, in particular, combs.

**Theorem 2.** DMVP for a single agent in $\mathcal{R}$ is NP-hard to approximate within any factor, even when the underlying graph is a comb.

**Proof.** Analogous to Theorem 1, we reduce from the set cover problem (SCP) [35]. Given an instance of SCP, construct a comb $G = (V, E)$ with backbone $b_0b_1b_{m+n+1}$; teeth $v_1, ..., v_m$, corresponding to elements in $\mathcal{U}$, with $(b_i, v_i) \in E \forall i = 1, ..., m$; teeth $p_1, ..., p_n$, corresponding to sets in $\mathcal{S}$, with $(b_{m+i}, p_i) \in E \forall i = 1, ..., n$; and two check teeth $p_0$ and $p_{n+1}$ with $(b_0, p_0), (b_{m+n+1}, p_{n+1}) \in E$. Let $B = \{(b_0, b_1), ..., (b_{m+n-1}, b_{m+n})\}$ be the set of all edges in the backbone of $G$. We use the following static subgraphs of $G$ to construct a TVG $\mathcal{G}$. For all $s_i$ in $\mathcal{S}$, let $pass(i) = (V, B \cup \{(b_{m+i}, p_i)\})$, and let $take(i) = (V, E_i)$, where $E_i = B \cup \{(b_j, v_j) : j \in s_i\}$. Let $check = (V, \{(b_0, p_0)\})$. Let $finish = (V, F)$, where $F = B \cup \{(b_{m+i}, p_i) : 1 \leq i \leq n + 1\} \cup \{(b_{m+n+1}, p_{n+1})\}$. Let $back = (V, B)$.

Define the TVG $\mathcal{G} = (back, m+n), (pass(1), 1), (take(1), 3m), (pass(1), 1), ..., (back, m+n)(pass(n), 1), (take(n), 3m), (pass(n), 1), (back, m+n), (check, 2), (finish, m+n+2+2k)$. The total duration of $\mathcal{G}$ is $D = n(4m+n+2) + (m+n) + 2 + (m+n+1+2k) = n^2 + 4mn + 4n + 2m + 2k + 3$.

Consider the problem of deciding if DMVP over $\mathcal{G}$ with a single agent $a$ starting at $b_0$ has a solution of length no more than $D$. This problem is in NP, since given a journey over $\mathcal{G}$, we can easily check that it hits every vertex, and that all of its edges are available at the correct times. Such a solution can be no longer than $D$, since $D$ is the total duration of $\mathcal{G}$.

Suppose $\mathcal{S}$ contains a cover $\mathcal{C}$ of $\mathcal{U}$ of size $k$ or less. Then for all $s_i \in \mathcal{C}$, $a$ takes $s_i$, that is, $a$ travels to $b_0$ during $back$, and visits all $v_j \in s_i$, and returns to the backbone during $take(i)$, which is possible since the duration of $take(i)$ is $3m$, ...
which allows $a$ to take all of the at most $m$ available elements while traveling up the backbone. Since $C$ is a cover of $\mathcal{U}$, $a$ visits all $v_i$. For all $s_i \notin C$, $a$ passes $s_i$, that is, $a$ moves to $b_{i+m}$ during back, and to $p_i$ during the first pass$(i)$, waits at $p_i$ during take$(i)$, and returns to $b_{i+m}$ during the second pass$(i)$. During the final back, $a$ moves to $b_0$, and during check, $a$ moves from $b_0$ to $p_0$, then back to $b_0$. At this point, since $|C| \leq k$, $a$ has passed at least $n - k s_i$'s. So, there are no more than $k p_i$’s left unvisited. $a$ visits these during finish, each at cost 2 off the path length $m + n + 2$ path up the backbone to $p_{n+1}$, thus completing visitation of all vertices of $G$ in no more than $D$ steps (e.g., Table 2).

Suppose there exists a solution to this instance of DMVP of length no more than $D$. Prior to finish, $a$ must have visited at least $n - k p_i$’s, since finish only lasts for $2k - 1$. So $a$ must have passed at least $n - k s_i$’s. Taking and passing for a single $s_i$ are mutually exclusive, because if $a$ moves to $p_i$ during the first pass$(i)$, $a$ must wait during take$(i)$, and if $a$ both takes $s_i$ and moves to $p_i$ during the second pass$(i)$, $a$ will be trapped at $p_i$ until finish, and will never be able to reach $p_0$, which must be visited during check, the only input time at which $p_0$ is available. Thus, $a$ could have taken no more than $k s_i$’s. During these $k$ or fewer takes, $a$ must have covered all $v_1, \ldots, v_m$. So, the union of these $k$ or fewer take$(i)$’s contains all edges $(c, v_j)$, which implies that the corresponding $s_i$’s form a cover of $\mathcal{U}$ or size $k$ or less. Hence, the decision problem is NP-complete.

Consider the minimization version of the problem with the same setup. Since it is NP-hard to decide if there is a solution of length $D$ or less, it NP-hard to find such a solution. But after $D$ steps, $a$ may have to wait an arbitrarily long time for the next edge is a feasible solution to appear, so any feasible solution that takes longer than $D$ steps can be arbitrarily long. Therefore, the problem cannot be approximated within any factor. \qed
Table 2. Thm.2 sample static snapshots for $U = \{1, 2, 3, 4, 5\}, s_1 = \{1, 2, 4\} \in S$, with $|S| = 4$, and $k = 2$.  

Agent location after each static temporal subgraph in dark gray; nodes covered so far in light gray.
3.2. \( B \): Time-bounded recurrent edges

We have a similar set of lower bounds for the case of \( B \), but with some ability to approximate. We later show (Theorem 13) that these approximation bounds are indeed tight for all trees.

**Theorem 3.** DMVP for a single agent in \( B \) is NP-hard to approximate within any factor less than \( \Delta \), even when the underlying graph is a spider, \( \forall \Delta > 1 \).

**Proof.** We reduce from 3-partition. Given a multiset of \( 3m \) positive integers \( S = \{s_1, ..., s_{3m}\} \), it is strongly NP-complete to decide if they can be partitioned into \( m \) sets where all sets have the same sum \( [29] \). Let \( \sum_{i=1}^{3m} s_i = M \). Then \( B = M/m \) is the required sum for each partition.

Starting with the common central vertex \( c \), construct a spider in the following way. For each \( s_i \in S \), add a corresponding arm of length \( s_i \). Add \( m \) arms of length one to be used as checkpoints, and add a single long arm \( A_0 \) of some length \( k > 2M + 2m \) (e.g., Figure 7). For the TVG used in this proof, arms over any period of time are in one of three modes: steady, flashing, or carrying. When an arm \( A \) is steady over a period of time from \( \tau \) to \( \tau' \), all of its edges are constantly available during that period. When \( A \) is flashing, all of its edges synchronously alternate between being unavailable for \( \Delta - 1 \) steps, and available for 1 step, satisfying the time-bounded recurrence constraint. Formally, if \( A \) is flashing, then \( \forall e \in A, t \in [\tau, \tau'] \),

\[
\rho(e, t) = \begin{cases} 
1 & \text{if } t - \tau \equiv p - 1 \text{ mod } p, \\
0 & \text{otherwise}.
\end{cases}
\]

When \( A \) is carrying, all of its edges act as if \( A \) were flashing, with the exception that the edge distance \( i \) from \( c \) is always available at time \( \tau + i \), so that starting at
time \( \tau \), an agent at \( c \) can travel along \( A \) for \( \tau' - \tau \) steps without waiting. Formally, if \( A = ca_0...a_l \) is carrying, then \( \forall (u,v) \in E(A), t \in [\tau, \tau'] \),

\[
\rho(e,t) = \begin{cases} 
1 & \text{if } t - \tau \equiv p - 1 \mod p, \\
1 & \text{if } v = a_{t-\tau}, \\
0 & \text{otherwise}.
\end{cases}
\]

Let \( \text{take} \) be the temporal subgraph of duration \( 2B \) in which the arms corresponding to \( s_i \)'s are steady, and all others are flashing. Let \( \text{check} \) be the temporal subgraph of duration \( 2 \) in which all checkpoint arms are steady, and all others are flashing. Let \( \text{finish} \) be the temporal subgraph of duration \( k \) in which all arms are flashing except for \( A_0 \), which is carrying. Let \( \mathcal{G} \) be the TVG formed by alternating between \( \text{take} \) and \( \text{check} \) \( m \) times, before ending with \( \text{finish} \). The total duration of \( \mathcal{G} \) is \( D = 2M + 2m + k \).

Consider the problem of deciding if DMVP over \( \mathcal{G} \) with a single agent \( a \) starting at \( c \) has a solution of length no more than \( D \). Since \( k > 2M + 2m \), such a solution must take the long arm last, as traversing this arm twice would result in a solution of length greater than \( 2k > (2M + 2m) + k = D \). Furthermore, since every arm must be traversed twice except the long arm, the topological length of a solution journey can be no less than \( 2M + 2m + k = D \). So a solution of temporal length no more than \( D \) cannot involve waiting.

Suppose there exists a 3-partition of \( S \). During each \( \text{take} \), \( a \) can explore the arms of the spider corresponding to one partition, and return to \( c \) in exactly \( 2B \) steps. During the subsequent \( \text{check} \), \( a \) visits one checkpoint arm, and returns to \( c \) in the allotted 2 steps. Repeating this process for the remaining \( \text{takes} \) and \( \text{checks} \), \( a \) will cover all the \( s_i \) arms and checkpoint arms without ever waiting, and end
up again at \( c \). Finally, during \textit{finish}, \( a \) takes the long arm \( A_0 \), reaching its leaf without waiting, completing coverage in \( D \) steps.

Now, suppose this instance of DMVP has a solution of length \( D \). To avoid waiting, \( a \) must take complete arms and return to \( c \) during each \textit{take}, so that it is not stalled by flashing. Similarly, \( a \) must take a single checkpoint arm and return to \( c \) during each \textit{check}. Doing this \( m \) times, \( a \) has effectively partitioned the \( s_i \) arms into sets each of total length \( B \). So, the decision problem is NP-complete, since 3-partition remains NP-complete even when input integers are given in unary. \( a \) completes the solution by following the long arm \( A_0 \), which can be traversed in \( k \) steps immediately after \( a \) returns to \( c \) from the final checkpoint.

Consider the minimization version of the problem with the same setup. Note that if \( a \) does not begin taking \( A_0 \) right as \textit{finish} begins, \( A_0 \) will take at least \( \Delta (k-1) + 1 \) to traverse, this best case occurring when \( a \) does not have to wait for the first edge. In particular, if \( a \) takes \( A_0 \) last, but has not visited all other arms before \textit{finish} starts, visiting those arms must have taken at least \( 2M + 2m + (\Delta - 1) > 2M + 2m, \forall \Delta > 1 \), since \( a \) must wait at least once during its traversal. The total cost of the solution is then at least \( D' = \Delta (k-1) + 1 + 2M + 2m + (\Delta - 1) = \Delta k + 2M + 2m \). If \( a \) does not take \( A_0 \) last, it must traverse \( A_0 \) twice, taking at least \( \Delta (k-1) + 1 + k \) steps (this best case occurring when \( a \) starts taking the long arm out right as \textit{finish} starts), and once it returns must wait at least once while traversing the remaining arms, making the length of the total solution at least \( D'' = \Delta (k-1) + 1 + k + 2M + 2m + (\Delta - 1) = \Delta k + 2M + 2m + k > D' \). Take any real constant \( \delta < \Delta \). Choose the least integer \( N \) s.t. \( N > \frac{1}{\Delta - \delta} \). Let \( k = N\delta (2M + 2m) \). Then,

\[
(\Delta - \delta)N\delta (2M + 2m) > \delta (2M + 2m),
\]
Therefore, any solution that contains waiting cannot be a $\delta$-approximation. So, finding a $\delta$-approximation is equivalent to finding a solution with no waiting, i.e., a minimal solution, and thus is NP-hard. Hence, the problem is NP-hard to approximate within any factor less than $\Delta$.

\[ (\Delta - \delta)k > \delta(2M + 2m), \]
\[ \Delta k > \delta(2M + 2m + k), \]
\[ \Delta k + 2M + 2m = D' > \delta D, \forall \Delta > 1. \]

Therefore, any solution that contains waiting cannot be a $\delta$-approximation. So, finding a $\delta$-approximation is equivalent to finding a solution with no waiting, i.e., a minimal solution, and thus is NP-hard. Hence, the problem is NP-hard to approximate within any factor less than $\Delta$.

**Theorem 4.** DMVP for a single agent in $\mathcal{B}$ is NP-hard to approximate within any factor less than $\Delta$, even when the underlying graph is a comb, $\forall \Delta > 1$.

**Proof.** We use a similar extension to that for $\mathcal{R}$ to extend this result from spider to a comb with long enough arms. We again reduce from 3-partition. Given a multiset of $3m$ positive integers $S = \{s_1, ..., s_{3m}\}$, it is strongly NP-complete to decide if they can be partitioned into $m$ sets where all sets have the same sum [29]. This result clearly still holds even when $m$ is even. So suppose $m$ is even and, let $\sum_{i=1}^{3m} s_i = M$. Then $B = M/m$ is the required sum for each partition.

Let $l = \frac{7m^2}{2} - \frac{3m}{2} + 4$. Starting with a backbone $\beta = b_1...b_{4m+1}$, construct a comb in the following way: For each $s_i \in S$, add a corresponding arm of length...
times $l_{s_i}$ attached to $\beta$ at $b_{\frac{2i}{2}+i}$. Add $m$ arms $c_1, \ldots, c_m$ of length one to be used as checkpoints, with $c_i$ attached at $b_{\frac{2i}{2}+\frac{1}{2}}$ if $i$ is odd, and $b_{\frac{2m}{2}+\frac{i}{2}}$ if $i$ is even. Add a single long arm of some length $k > 2lBm + \frac{7m^2}{2} - \frac{2m}{2} + 3$, attached at $b_{4m+1}$. For the TVG used in this proof (e.g., Figure 8), arms over any period of time are in one of three modes: steady, flashing, or carrying (see the proof of Theorem 3).

Assume all edges in $\beta$ be available at all times, unless stated otherwise. Let $\text{take}$ be the temporal subgraph of duration $2lB + 3m$ in which the arms corresponding to $s_i$'s are steady, and all others are flashing. Let $\text{check}(j)$ be the temporal subgraph of duration $i + (i \mod 2) + 2$ in which checkpoint $c_j$'s arm is steady, and all others are flashing. Let $\text{finalcheck}$ be the temporal subgraph of duration $\frac{m}{2} + 2$ in which checkpoint $c_m$'s arm is steady, and all others are flashing. Let $\text{finish}$ be the temporal subgraph of duration $k$ in which $\beta$ is flashing, and all arms are flashing except for the long arm of length $k$, which is carrying. Let $G$ be the TVG $\text{take, check}(1), \text{take, check}(2), \ldots, \text{take, check}(m-1), \text{take, finalcheck, finish}$. The duration of $G$ up until the start of $\text{finish}$ is $d = 2lBm + 3m^2 + \sum_{i=1}^{m}(i + (i \mod 2) + 2) + \frac{m}{2} + 3 = 2lBm + \frac{7m^2}{2} - \frac{2m}{2} + 3$. The total duration of $G$ is $D = d + k$.

Consider the problem of deciding whether DMVP over $G$ with a single agent $a$ starting at $b_{\frac{2m}{2}}$ has a solution of length no more than $D$. Since $k > 2lBm + \frac{7m^2}{2} - \frac{2m}{2} + 3$, such a solution must take the long arm last, as traversing this arm twice would result in a solution of length greater than $2k > (2lBm + \frac{7m^2}{2} - \frac{2m}{2} + 3) + k = D$.

Suppose there exists a 3-partition of $S$. During the $j$th $\text{take}$, if $j$ is odd (even), starting at $b_{\frac{2m}{2}} (b_{\frac{2m+1}{2}})$, $a$ can explore the arms of the spider corresponding to one partition, and end up at $b_{\frac{2m}{2}+1} (b_{\frac{2m+1}{2}})$ in exactly $2Bl + 3m$ steps. During the subsequent $\text{check}(j)$, $a$ visits $c_j$, and returns to $b_{\frac{2m}{2}+1} (b_{\frac{2m}{2}})$ in exactly $i + \ldots$
(i mod 2) + 2 steps. During finalcheck, a travels from $b_{\frac{m}{2}}$ to $c_m$ and then to $b_{4m+1}$ in the allotted $\frac{m}{2} + 2$ steps. Finally, during finish, a takes the long arm, reaching its leaf without waiting, completing coverage in $D$ steps.

Now, suppose this instance of DMVP has a solution of length $D$. If there is no 3-partition of $S$, then a must wait during traversal of at least one arm corresponding to an $s_i$. Since the length of this arm is $l_{s_i}$, a must in fact wait for at least $l$ edges of this arm, incurring a cost of $(\Delta - 1)l = (\Delta - 1)(\frac{7m^2}{2} - \frac{3m}{2} + 2)$. Since in the length $D$ solution described above a never waits on an arm, this incurred cost must be made up for by minimizing traversal of edges in $\beta$. However, in the length $D$ solution described above, a only traverses $\beta$ for a total of $\frac{7m^2}{2} - \frac{3m}{2} + 1 < (\Delta - 1)l$ steps, $\forall \Delta > 1$. Therefore, the solution must be in the form of the solution described above in which a 3-partition of $S$ does indeed exists. So, the decision problem is NP-complete, since 3-partition remains NP-complete even when input integers are given in unary.

Consider the minimization version of the problem with the same setup. Note that if a does not begin taking the long arm right as finish begins, the long arm will take at least $\Delta(k - 1) + 1$ to traverse, this best case occurring when a does not have to wait for the first edge. In particular, if a takes the long arm last, but has not visited all other arms before finish starts, visiting those arms must have taken at least $d + (\Delta - 1)$, since a must wait at least once during their traversal, and the total cost of the solution is then at least $D' = \Delta(k - 1) + 1 + d + (\Delta - 1) = \Delta k - 1 + d$. If a does not take the long arm last, it must traverse the long arm twice, taking at least $\Delta(k - 1) + 1 + k$ steps (this best case occurring when a starts taking the long arm right as finish starts), and once it returns must wait at least once while traversing the remaining arms, making the length of the total solution at least $D'' = \Delta(k - 1) + 1 + k + d + (\Delta - 1) = \Delta(k + 1) + d > D'$. Take any real constant
\[ \delta < \Delta. \] Choose the least integer \( N \) s.t. \( N > \frac{1}{\Delta - \delta} \). Let \( k = N\delta d \). Then

\[ (\Delta - \delta)N\delta d > \delta d, \]
\[ (\Delta - \delta)k > \delta d, \]
\[ \Delta k > \delta d + \delta k, \]
\[ \Delta k + d = D' > \delta D, \forall \Delta > 1. \]

Therefore, any solution that contains waiting cannot be a \( \delta \)-approximation. So, finding a \( \delta \)-approximation is equivalent to finding a solution with no waiting, i.e., a minimal solution, and thus is NP-hard. Hence, the problem is NP-hard to approximate within any factor less than \( \Delta \). \qed

### 3.3. \( \mathcal{P} \): Periodic edges

As is shown in Chapter 4, there is a much greater potential for tractability of DMVP in \( \mathcal{P} \) than in \( \mathcal{B} \) or \( \mathcal{R} \). However, the next result follows immediately via reduction from the hamiltonian path problem by simply restricting \( t \) to \( n - 1 \). 
3.3. \( \mathcal{P} \): PERIODIC EDGES

**Theorem 5.** DMVP for a single agent in \( \mathcal{P} \) is NP-complete, when \( p = 1 \).

**Proof.** \( p = 1 \) is simply the static case, so the theorem follows immediately from the result that MVP is NP-complete for a single agent on general graphs [1]. \( \square \)

DMVP in \( \mathcal{P} \) for \( p = 1 \) is then also NP-complete for all classes of graphs for which HAM-PATH is NP-Hard, in particular, planar graphs of maximum degree 3, bridgeless undirected planar 3-regular bipartite graphs, and 3-connected 3-regular bipartite graphs [5]. To show that \( \mathcal{P} \) is an interesting dynamics class for DMVP, it is important to show that DMVP yields different hardness results over \( \mathcal{P} \) than over static graphs. Thus, we construct a class of graphs for the following result:

**Theorem 6.** There is a class of graphs \( C \) such that DMVP for a single agent in \( \mathcal{P} \) over graphs in \( C \) is NP-complete when \( p = 2 \), but trivial when \( p = 1 \).

**Proof.** Given a graph \( G \) with an even number of vertices arbitrarily ordered \( v_0, ..., v_{n-1} \), construct a corresponding graph \( G' \in C \) by adding \( n \) new vertices \( c_0, ..., c_{n-1} \), and adding the edges \((v_i, c_i), (v_i, c_{i+1})\), and \((c_i, c_{i+1})\) for all \( 0 < i < n \), where indices are taken mod \( n \).

To show the problem is NP-complete for a single agent in \( \mathcal{P} \) over graphs in \( C \), with \( p = 2 \), we reduce from the hamiltonian path problem [35]. Consider a graph \( G \) with an even number of vertices \( n \), and one of those vertices \( v_0 \), with the problem of deciding whether \( G \) contains a hamiltonian path starting at \( v_0 \). Take the graph \( G' \in C \) corresponding to \( G \) as the underlying graph of \( G \). \( G \) begins at time 0. In \( \mathcal{P} \) with \( p = 2 \), traversable edges can only be one of three possible types: (01) available at odd times but not even times, (10) available at even times but not odd times, (11) available at all times. Let all original edges of \( G \) be of type 11. Let \((v_i, c_i)\) be of type 01 when \( i \) is even and type 10 when \( i \) is odd. Let
(v\_i, c\_i+1) be of type 10 when \(i\) is even and 01 when \(i\) is odd. Let \((c\_i, c\_i+1)\) be of type 10 when \(i\) is even and 01 when \(i\) is odd (see Figure 9).

Consider the problem of deciding if DMVP over \(G\) for a single agent \(a\) starting at \(v_0\) has a solution of length no more than \(2n - 1\), i.e., a solution with no waiting, and in which each vertex is visited exactly once. This problem is clearly in NP. If there is a hamiltonian path in \(G\) from \(v_0\) vertex \(v_i\), then this path will be constantly available in \(G\). \(a\) can take this path in \(n - 1\) steps, and, ending at an odd time, immediately follow \((v_i, c_i)\) if \(i\) is even, or \((v_i, c_{i+1})\) if \(i\) is odd, then follow either the path \(c_i c_{i+1} c_{i+2} \ldots c_{i-1}\) or \(c_{i+1} c_{i+2} c_{i+2} \ldots c_i\), the edges for which are always available as \(a\) reaches the incoming vertices, thus completing the overall traversal in exactly \(2n - 1\) steps. Suppose there is a solution to this problem of length \(2n - 1\). By construction, if \(a\) moves to any \(c_i\) before covering every \(v_j\), \(a\) must then wait at least once at some \(c_k\) before visiting any further \(v_l\). This is because for all \(c_i\), once \(c_i\) is reached via either \((v_{i-1}, c_i)\), \((v_i, c_i)\), or \((c_{i-1}, c_i)\) the only edge that can be immediately taken without waiting is \((c_i, c_{i+1})\). So \(a\) must visit all \(v_j\) exactly once without visiting any \(c_i\), thus following a path corresponding to a hamiltonian path in \(G\).

However, if we consider the same setup over \(G'\) but with \(p = 1\), \(v_0 c_1 v_1 c_2 v_2 \ldots c_{n-1} v_{n-1} c_0\) is always an optimal solution. □
Figure 9. $G$ (from Thm. 6) with underlying graph $G'$ constructed from some six-node graph $G$. Edges labeled 10 are available at even times; 01 at odd times. Edges in $G$ are available at all times.
CHAPTER 4

Upper Bounds

4.1. $\mathcal{R}$: Recurrent edges

In this section, we map out a class of graphs over which DMVP in $\mathcal{R}$ is solvable in polynomial time. We first start with a very useful lemma. Note that a related observation (specifically, about turning around on a ring) was made in [33].

**Lemma 1 (Turning around lemma).** There is always an optimal solution $J$ that never turns around at a degree 2 vertex of the edge-induced subgraph of $J$ in $G$.

**Proof.** Suppose $v$ is a degree 2 vertex with neighbors $u, w$ in the edge-induced subgraph of $J$ in $G$. Suppose agent $a$ takes edge $(u, v)$ at time $\tau$, then turns around, taking $(u, v)$ at time $\tau'$ as the very next edge in its traversal. Since $(v, w)$ is in the edge-induced subgraph of $J$, $a$ must traverse $(v, w)$ at some other time, thereby reaching $v$ at that time. So, $a$ could have waited at $u$ from times $\tau$ to $\tau' + 1$, instead of taking $(u, v)$ at time $\tau$, and the solution would still be optimal. \qed

We apply Lemma 1 to get the following solvability results for restricted classes of underlying graphs. DMVP over a path or a cycle is motivated in part by coverage of a dynamic border.

**Theorem 7.** DMVP for a single agent in $\mathcal{R}$ on a path is solvable in $O(T)$ time.

**Proof.** Consider DMVP for a single agent $a$ with underlying graph $G$ the path $v_0v_1...v_n$, and $a$ starting at $v_k$. To reach $v_0$, $a$ must cover all $v_{k-1}, ..., v_1$ along
the way. Similarly, to reach \( v_n \), \( a \) must cover all \( v_{k+1}, \ldots, v_{n-1} \). By Lemma 1, an optimal solution can be constructed by first taking a foremost journey to either \( v_0 \) or \( v_n \), then taking the foremost journey to the remaining endpoint. One of these two topological journeys, called the left and right journeys, must embody an optimal solution, but in the worst case edge availability must be checked for all \( t \in T \), yielding an \( O(T) \) runtime. For the sake of completeness, we give pseudocode for the algorithm below (Algorithm 3).

**Algorithm 3 DMVP-Line(\( G, \{v_k\} \))**

\[
\begin{align*}
&lLoc = rLoc = k \\
&lTurned = rTurned = complete = False \\
&t = 0 \\
\textbf{while} \not\textbf{complete} \textbf{do} \\
&\text{if not} \ lTurned \textbf{then} \\
&\quad \text{if} \ \rho((v_{lLoc}, v_{lLoc-1}), t) = 1 \textbf{then} \\
&\quad \quad lLoc = lLoc - 1 \\
&\quad \text{if} \ lLoc = 0 \textbf{then} \\
&\quad \quad lTurned = True \\
&\quad \text{else} \\
&\quad \quad \text{if} \ \rho((v_{lLoc}, v_{lLoc+1}), t) = 1 \textbf{then} \\
&\quad \quad \quad lLoc = lLoc + 1 \\
&\quad \quad \text{if} \ rLoc = n \textbf{then} \\
&\quad \quad \quad complete = True \\
&\text{if not} \ rTurned \textbf{then} \\
&\quad \text{if} \ \rho((v_{rLoc}, v_{rLoc+1}), t) = 1 \textbf{then} \\
&\quad \quad rLoc = rLoc + 1 \\
&\quad \text{if} \ rLoc = n \textbf{then} \\
&\quad \quad rTurned = True \\
&\text{else} \\
&\quad \text{if} \ \rho((v_{rLoc}, v_{rLoc-1}), t) = 1 \textbf{then} \\
&\quad \quad rLoc = rLoc - 1 \\
&\quad \text{if} \ rLoc = 0 \textbf{then} \\
&\quad \quad complete = True \\
&t = t + 1 \\
\textbf{return} \ t
\end{align*}
\]
Using dynamic programming, we also get an efficient algorithm for \( k \) agents on a path, whose runtime is independent of \( k \).

**Theorem 8.** DMVP for \( k \) agents in \( \mathcal{R} \) on a path is solvable in \( O(Tn) \) time.

**Proof.** Consider DMVP with underlying graph the path \( P = v_1...v_n \) and \( k \) agents \( a_1, ..., a_k \) starting at locations \( s_1, ..., s_k \), respectively. Suppose \( P \) is oriented left to right, with \( v_1 \) the leftmost vertex. Without loss of generality, suppose \( s_1, ..., s_k \) are ordered from left to right. The idea here is to compute for each vertex \( u \in s_1...v_n \) the DMVP subproblem over \( v_1...u \) for all agents starting on or to the left of \( u \). Call this value \( c(u) \). We can compute this from left to right, and finally get the result \( c(v_n) \) for DMVP for all \( k \) agents over \( P \) (Algorithm 4).

Note that if two or more agents start at the same vertex, simply sending two of them in opposite directions will be trivially optimal, so assume no two agents start at the same node. On a path, it is never advantageous for any two agents to cross over one another, since they could simply each turn around instead. As a result, agent \( a_1 \) must cover \( v_1 \). Let \( v \) be the node directly to the left of \( s_2 \). The subproblems to be computed from \( c(s_1) \) to \( c(v) \) concern only agent \( a_1 \). \( c(s_1) \) is the time it takes \( a_1 \) to reach \( v_1 \) by simply traveling left starting at time 0. For all \( u \) strictly between \( s_1 \) and \( s_2 \), \( a_1 \) can cover \( v_1...u \) either by going left first or right first. We can compute all left-first journeys in a single pass in \( O(T) \) by going left until hitting \( v_1 \), then turning around and recording the time at which each \( u \) is reached. For the journeys that go right first, \( a_1 \) travels right to \( u \), turns around and travels left until \( v_1 \) is reached. For each \( u \), the minimum of the left-first and right-first journey is stored as \( c(u) \). Doing this for each \( u \) takes overall \( O(T\|s_1...s_2\|) \).

Now consider any agent \( a_i \) in \( \{a_2, ..., a_{k-1}\} \), and suppose all subproblems to the left of \( s_i \) have already been computed. Let \( L_i \) be the path from the right neighbor of \( s_{i-1} \) to \( s_i \), and \( R_i \) be the path from \( s_i \) to the left neighbor of \( s_{i+1} \). In
a full optimal solution over $P$, the leftmost vertex $a_i$ covers could be any vertex in $L_i$, and the rightmost vertex could be any in $R_i$. $c(s_i)$ is the minimum over all $v_j$ in $L_i$, of the maximum of $c(v_{j-1})$ and the time it takes $a_i$ to reach $v_j$ traveling left from time 0. This is computed in a single $O(T)$ left pass. Now suppose the rightmost vertex $a_i$ covers is not $s_i$. Then, if $a_i$ goes left first and turns around at $l$, we can compute the cost of $a_i$’s journey ending at each vertex $r \neq s_i$ in $R_i$ in a single $O(T)$ pass, in which $a_i$ turns around at $l$ and then travels right as far as possible. Doing this for each $l$ takes overall $O(T|L_i|)$. Similarly, if $a_i$ goes right first and turns around at $r$, we can compute the cost of $a_i$’s journey ending at each vertex $l \neq s_i$ in $L_i$ in a single $O(T)$ pass, in which $a_i$ turns around at $r$ and then travels left as far as possible. Doing this for each $r$ takes overall $O(T|R_i|)$. For each $r \in R_i$, $c(r)$ is the minimum over all $v_j \in L_i$, of the maximum of $c(v_{j-1})$ and the minimum between the left-first and right-first journeys of $a_i$ covering $v_j...r$. $c(r)$ can simply be updated immediately anytime a better solution is evaluated.

$a_k$ faces a similar situation to $a_1$, it must cover $v_n$, so only needs to consider variable left endpoints. The cost of the optimal solution over all of $P$ is then the minimum over all $v_j \in L_k$ of the max of $c(v_{j-1})$ and the minimum between the left-first and right-first journeys of $a_k$ covering $v_j...v_n$. Computation of the complete DMVP solution over $P$ takes $O(T|R_1|) + O(T|L_2|) + O(T|R_2|) + ... + O(T|L_{k-1}|) + O(T|R_{k-1}|) + O(T|L_k|) = O(Tn)$. □

We have a similar pair of efficient upper bounds for the cycle.

**Theorem 9.** DMVP for a single agent in $\mathcal{R}$ on a cycle is solvable in $O(Tn)$ time.

**Proof.** A similar case to Theorem 7 can be made for the cycle $C = v_0v_1...v_nv_0$. Suppose $a$ starts at $v_0$ at time 0. Consider an optimal visitation of $C$ for $a$. In this optimal solution, there is some vertex $v_k \neq v_0$ that is visited last. The second to
Algorithm 4 DMVP-k-Line(\(P, \{s_1, \ldots, s_k\}\))

\[
\text{for all } v \in s_1 \ldots v_n \text{ do} \quad \triangleright \text{Initialize } c \\
\quad c(v) = \infty \\
\text{for } i = 1, \ldots, k \text{ do} \\
\quad lBoundary = s_i \quad \triangleright \text{Evaluate all left-first journeys for } a_i \\
\quad \text{if } i = 1 \text{ then} \\
\quad \quad lBoundary = v_1 \\
\quad \text{while } lBoundary \notin \{s_{i-1}, \emptyset\} \text{ do} \quad \triangleright \text{Try every possible left endpoint} \\
\quad \quad t = 0 \\
\quad \quad \text{loc} = s_i \\
\quad \quad \text{turned} = \text{eval} = \text{False} \quad \triangleright \text{evaluate solution?} \\
\quad \quad \text{while } \text{loc} \notin \{\emptyset, s_{i+1}\} \text{ and } t < T \text{ do} \quad \triangleright \text{enter at most } T \text{ times} \\
\quad \quad \quad \text{if } \text{loc} = \text{lBoundary} \text{ then} \\
\quad \quad \quad \quad \text{turned} = \text{True} \\
\quad \quad \quad \quad \text{if } \text{turned} = \text{True} \text{ and } \text{loc} = s_i \text{ then} \\
\quad \quad \quad \quad \quad \text{eval} = \text{True} \\
\quad \quad \quad \quad \text{if } \text{eval} = \text{True} \text{ then} \\
\quad \quad \quad \quad \quad c(\text{loc}) = \min(c(\text{loc}), \max(c(\text{lBoundary.lNode}), t)) \\
\quad \quad \quad \quad \text{if not } \text{turned} \text{ and } \rho(\text{loc.lEdge}, t) = 1 \text{ then} \\
\quad \quad \quad \quad \quad \text{loc} = \text{loc.lNode} \\
\quad \quad \quad \quad \text{if } \text{turned} \text{ and } \rho(\text{loc.rEdge}, t) = 1 \text{ then} \\
\quad \quad \quad \quad \quad \text{loc} = \text{loc.rNode} \\
\quad \quad \quad \quad t = t + 1 \\
\quad \quad lBoundary = \text{lBoundary.lNode} \\
\quad \quad \text{rBoundary} = s_i \quad \triangleright \text{Evaluate all right-first journeys for } a_i \\
\quad \quad \text{if } i = k \text{ then} \\
\quad \quad \quad \text{rBoundary} = v_n \\
\quad \quad \text{while } \text{rBoundary} \notin \{s_{i+1}, \emptyset\} \text{ do} \quad \triangleright \text{Try every possible right endpoint} \\
\quad \quad \quad t = 0 \\
\quad \quad \quad \text{loc} = s_i \\
\quad \quad \quad \text{turned} = \text{eval} = \text{False} \quad \triangleright \text{evaluate solution?} \\
\quad \quad \quad \text{while } \text{loc} \notin \{s_{i-1}, \emptyset\} \text{ and } t < T \text{ do} \quad \triangleright \text{enter at most } T \text{ times} \\
\quad \quad \quad \quad \text{if } \text{loc} = \text{rBoundary} \text{ then} \\
\quad \quad \quad \quad \text{turned} = \text{True} \\
\quad \quad \quad \quad \text{if } \text{turned} = \text{True} \text{ and } \text{loc} = s_i \text{ then} \\
\quad \quad \quad \quad \quad \text{eval} = \text{True} \\
\quad \quad \quad \quad \text{if } \text{eval} = \text{True} \text{ then} \\
\quad \quad \quad \quad \quad c(\text{rBoundary}) = \min(c(\text{rBoundary}), \max(c(\text{loc.lNode}), t)) \\
\quad \quad \quad \quad \text{if not } \text{turned} \text{ and } \rho(\text{loc.rEdge}, t) = 1 \text{ then} \\
\quad \quad \quad \quad \quad \text{loc} = \text{loc.rNode} \\
\quad \quad \quad \quad \text{if } \text{turned} \text{ and } \rho(\text{loc.lEdge}, t) = 1 \text{ then} \\
\quad \quad \quad \quad \quad \text{loc} = \text{loc.lNode} \\
\quad \quad \quad \quad t = t + 1 \\
\quad \quad \text{rBoundary} = \text{rBoundary.rNode} \\
\text{return } c(v_n)
last vertex is then either \( v_{k-1} \) or \( v_{k+1} \). If it is \( v_{k-1} \), then \( a \) must have already visited \( v_{k+1} \) without visiting \( v_k \). So, the edge \((v_{k+1}, v_k)\) is never traversed. Therefore, the solution reduces to an optimal solution over the path graph \( v_k v_{k-1} \ldots v_{k+1} \) starting at \( v_0 \). Similarly, if instead \( v_{k+1} \) is the vertex visited second to last, then \( a \) must have already visited \( v_{k-1} \) without visiting \( v_k \), and the solution reduces to an optimal solution over the path \( v_{k-1} v_{k-2} \ldots v_{k+1} v_k \). Since there are \( n-1 \) possible final vertices for an optimal solution, the cost of an optimal solution can be computed by for each of these vertices computing the minimal cost between optimal coverage of each of the two corresponding paths using Algorithm 3, and taking the minimum over all \( n-1 \) vertices possible final vertices (see Figure 10). This yields an \( O(Tn) \) runtime.

\[ \square \]

We can again extend this to the case of \( k \) agents.

**Theorem 10.** DMVP for \( k \) agents in \( \mathcal{R} \) on a cycle is solvable in \( O(Tn^2/k) \) time.

**Proof.** Consider DMVP over the cycle \( C = v_0 v_1 \ldots v_n v_0 \) for \( k \) agents \( a_1, \ldots, a_k \) ordered clockwise around the cycle at locations \( s_1, \ldots, s_k \), respectively. If any two agents start at the same node, then sending them in opposite directions will be optimal, and removing the segment they cover results in a subproblem over a path that can be solved with Algorithm 4 in \( O(Tn) \). If no two agents start at the same node, let \( d \) be the shortest distance between any two agents \( a_i, a_{i+1} \). Since there are \( k \) agents, \( d = O(n/k) \). The furthest that \( a_{i+1} \) covers counter-clockwise can be any node from \( s_{i+1} \) to the immediate clockwise neighbor of \( s_i \). For each of these \( O(n/k) \) potential left endpoints \( v_j \), we can run Algorithm 4 on the path consisting of \( C \) with the edge \((v_{j-1}, v_j)\) removed. Taking the minimum over all \( v_j \) results in an \( O(Tn^2/k) \) runtime. \( \square \)
Figure 10. The 8 possible underlying walks of solutions, satisfying Lemma 1, to the 5-cycle starting at $v_0$.

Now we show that despite the severe inapproximability of DMVP over $\mathcal{R}$, we can always compute an optimal solution in exponential time.

**Theorem 11.** DMVP for a single agent in $\mathcal{R}$ is solvable in $O(Tn^3 + n^2 2^n)$ time.

**Proof.** The proposed algorithm first computes all-pairs-all-times-foremost-journey of input TVG $\mathcal{G}$, using a straightforward dynamic programming algorithm, then uses this information to run another dynamic programming algorithm conceived along the lines of a standard method for TSP [10]. (We note that an algorithm for computing all single source foremost journeys from a given start time is given in [51], though in the context of a different dynamic graph model.)

Let $d^t_{uv}$ be the length of the foremost temporal journey from $u$ to $v$, starting at time $t$. The following algorithm (Algorithm 5) computes $d^t_{uv}$ for all vertex pairs $(u,v)$, and times $t \in \mathcal{T}$ for a given TVG $\mathcal{G}$:
Algorithm 5 All-pairs-all-times-foremost-journey($\mathcal{G}$)

\begin{algorithm}
\begin{algorithmic}
\FORALL{$u, v \in V \times V$}
\IF{$u = v$}
\STATE $d^T_{uv} = 0$
\ELSE
\STATE $d^T_{uv} = \infty$
\ENDIF\STATE $\triangleright$ Since input ends at $T$, agent cannot move.
\ENDFOR
\FOR{$t = T - 1, \ldots, 0$}
\FORALL{$u, v \in V \times V$}
\IF{$u = v$}
\STATE $d^t_{uv} = 0$
\ELSE
\STATE $d^t_{uv} = d^{t+1}_{uv} + 1$
\STATE $\triangleright$ In worst case, just wait at $u$.
\FORALL{$k \in V$}
\IF{$\rho((u, k), t) = 1$}
\STATE $d^t_{uv} = \min(d^t_{uv}, d^{k+1}_{kv} + 1)$
\STATE $\triangleright$ Check for better route.
\ENDIF\ENDFOR\ENDFOR
\ENDFOR
\end{algorithmic}
\end{algorithm}

At all times $t$, for all vertices $u \in V$, $d^t_{uu}$ is clearly 0. At time $T$, the time limit has been reached, so an agent cannot move to another vertex in any guaranteed time, and thus we set $d^T_{uv} = \infty$ for all $u \neq v$. For all $T - 1 \geq t \geq 0$, in the worst case an agent can wait at $u$ for one step, and take the foremost journey to starting at time $t + 1$. If there is a better journey than this, it must consist of not waiting, rather taking one of the edges available at time $t$ from $u$ to some vertex $k$. Subsequently taking the foremost journey from $k$ to $v$ starting at time $t + 1$ results in an optimal journey through $k$. Algorithm 5 clearly runs in $O(Tn^3)$ time, and uses $O(Tn^2)$ space.

The next algorithm (Algorithm 6) uses the $d^t_{uv}$ values computed by Algorithm 5 to compute the cost of a minimal solution to DMVP for a single agent in $\mathcal{R}$. Let $V' \subseteq V$ and $c(V', v)$ be the minimal time it takes to visit all vertices in $V'$ starting at vertex $s$ at time 0 and ending at vertex $v \in V'$.

After initializing the minimal costs for visiting subsets up to size 2, the algorithm repeatedly uses the minimal costs for size $i$ subsets to calculate $(V', v)$
Algorithm 6 \( DMVP(\mathcal{G}, \{s\}) \)

\[
\begin{align*}
c(\{s\}, s) &= 0 \quad \triangleright \text{Initialize subset of size 1.} \\
\text{for all } v \neq s \in V &\text{ do} \quad \triangleright \text{Initialize subsets of size 2.} \\
c(\{s, v\}, v) &= d_{sv} \\
\text{for } i = 3, \ldots, n &\text{ do} \quad \triangleright \text{Build up to subsets of size } n. \\
&\quad \text{for all } S \subseteq V \text{s.t. } |S| = i \text{ do} \\
&\quad &\quad \text{for all } v \neq s \in V \\
&\quad &\quad &\quad c(V', v) = \min_{u \neq s \in V' \setminus \{v\}} \left( c(V' \setminus \{v\}, u) + d_{uv}^{c(V' \setminus \{v\}, u)} \right) \\
&\quad &\quad \text{return } \min_{v \neq s \in V} (c(V, v))
\end{align*}
\]

for each size \( i + 1 \) subset \( V' \) and \( v \neq s \in V' \). Once computed up to size \( n \), the algorithm returns the minimal cost among journeys that cover all vertices. This is an optimal solution to DMVP because it is the minimum cost of taking foremost journeys between vertices that results in a complete cover. There are \( 2^n \) subsets of \( V \), and so \( n2^n \) subset-vertex pairs of the form \( (V', v) \). For each of these, the algorithm computes the minimum of \( O(n) \) values. So, Algorithm 6 has running time \( O(n^22^n) \). Since it saves one cost for each subset-vertex pair, Algorithm 6 also uses \( O(n^22^n) \) space. Sequentially running Algorithm 5 followed by Algorithm 6, we have a complete algorithm for DMVP for a single agent in \( \mathcal{R} \), with combined running time \( O(Tn^3 + n^22^n) \).

We use Theorem 11 to generalize Theorems 7 and 9 with the following:

**Theorem 12.** \( DMVP \) for a single agent in \( \mathcal{R} \) is fixed parameter tractable, when \( G \) is an \( m \)-leaf \( c \)-almost-tree, for fixed parameter \( m \), and \( c \) constant.

**Proof.** First, consider the restricted case where \( G \) is an \( m \)-leaf tree. Since every leaf must be visited, and visiting all leaves implies coverage of the entire tree, there is a minimal solution that can be thought of as an ordering of the set of leaves of \( G \), and the foremost journeys between them. In this case, there is only one way to visit any node, namely, on the way to a leaf. Using this observation
and Algorithm 6 from the proof of Theorem 11, we see that we only need to consider all orderings of leaves, instead of all orderings of vertices, yielding a run time of $O(Tn^3 + m^22^m)$, which is indeed fixed parameter tractable for parameter $m$.

Suppose the underlying graph $G$ of $G$ is an $m$-leaf $c$-almost-tree. Consider all edges $e$ such that removing $e$ from $G$ results in a $(c - 1)$-almost-tree. Each of these edges lies on some path $P$ such that removing any edge of $P$ will similarly result in a $(c - 1)$-almost-tree, and every intermediate vertex on the path has degree 2 (Figure 11). Suppose $P$ is the path $v_0...v_l$. Since $G$ is an $m$-leaf $c$-almost-tree, there are $O(m)$ paths of this type. The edge-induced subgraph $G'$ of the underlying walk of an optimal covering of $G$ can be any $(c - c')$-almost-tree $⊆ G$, for $0 ≥ c' ≥ c$. For each $c'$, a solution involves selecting $c'$ paths, each of $O(n)$ length, from which to remove an edge. So, there are $O(m^cn^c)$ possible choices of $(c - c')$-almost-trees, and thus $O(\sum_{c'=0}^{c}(m^cn^c)) = O(m^cn^c)$ choices for $G'$. Every $G'$ has no more than $m + 2c$ leaves. Since every edge of $G'$ is covered, by Lemma 1, there are at most 2 ways to cover each of the remaining $O(m)$ paths $v_0...v_l$ of intermediate vertex degree 2, namely: entering at $v_0$ and exiting at $v_l$, or entering at $v_l$ and exiting at $v_0$. Augment the set of leaves to be ordered in a solution with the selected ways of covering these paths, that is, select one of the consecutive subsequences $v_0v_l$ or $v_lv_0$ to be in the ordering. With this augmentation, we still have a set of $O(m)$ elements to be ordered, the optimal ordering of which can be computed via Theorem 11 in $O(Tn^3 + m^22^m)$ time. The minimum over all ways of covering $G'$ can then be computed in $O(2^m)O(Tn^3 + m^22^m) = O(Tn^32^m + m^22^m)$. The overall minimum cost for covering $G$ can then be computed by taking the minimum cost over all $O(c^mn^c)$ edge-induced subgraphs in $O(m^cn^c)O(Tn^32^m + m^22^m) = O(Tn^{3+c}f(m))$ time. □
Figure 11. The 7 possible ways, satisfying Lemma 1, of covering a length 5 path with degree 2 intermediate nodes.

The following result follows immediately for the case when $m = O(\lg n)$.

**Corollary 1.** DMVP for a single agent in $\mathcal{R}$ is solvable in polynomial time, if $G$ is an $O(\lg n)$-leaf $c$-almost-tree, for $c$ constant.

We conjecture (see Chapter 5) that the maximal class of graphs over which DMVP in $\mathcal{R}$ is poly-time solvable is the class of all graphs with polynomially many spanning trees, all of which have $O(\lg n)$ leaves. Theorem 12 also generalizes to the case of $k$ agents:

**Corollary 2.** DMVP for $k$ agents in $\mathcal{R}$ is fixed parameter tractable, when $G$ is an $m$-leaf $c$-almost-tree, for fixed parameters $m$ and $k$, and $c$ constant.

**Proof.** From the proof of Theorem 12, we know an optimal solution consists of a partition of $O(m)$ elements into $k$ sets, each of which is covered optimally by a single agent. There are $O(k^m)$ such partitions, so running Algorithm 6 for each agent for each partition takes $O(Tn^{3+c}k^{m+1}f(m)) = O(Tn^{3+c}f(m, k))$ time. □
4.2. \( \mathcal{B} \): Time-bounded recurrent edges

In \( \mathcal{B} \), we can tighten the upper bound from Theorem 11:

**Corollary 3.** DMVP for a single agent in \( \mathcal{B} \) is solvable in \( O(\Delta n^4 + n^2 2^n) \) time.

**Proof.** Since DMVP in \( \mathcal{B} \) is bounded by \( 2\Delta n \), no times later than this need to be considered. Thus, for \( \mathcal{B} \), the running time of the algorithm in Theorem 11 reduces to \( O(\Delta n^4 + n^2 2^n) \).

We also see that we are able to greatly improve on approximation from \( \mathcal{R} \) to \( \mathcal{B} \):

**Theorem 13.** DMVP for a single agent in \( \mathcal{B} \) over a tree can be \( \Delta \)-approximated in \( O(n) \) time. This approximation is tight.

**Proof.** In [1], it is shown that minimal MVP cost \( C \) can be computed in \( O(n) \) for static graphs. In the dynamic case, the journey corresponding to following exactly the edges in the static solution when they are available can be followed, waiting at most \( \Delta - 1 \) steps for each edge to appear before it is traversed. Since no solution can be better than \( C \), and the proposed journey takes at most \( \Delta C \), it must be a \( \Delta \)-approximation. From Theorems 3 and 4, we know there can be no better approximation. Hence, this approximation is tight.

**Theorem 14.** DMVP for a single agent in \( \mathcal{B} \) can be \( 2\Delta \)-approximated by any online spanning tree traversal of \( G \).

**Proof.** The topological length of a spanning tree traversal is no more than \( 2n - 3 \). In the worst case, waiting \( \Delta - 1 \) time steps for each subsequent edge to appear results in coverage of \( G \) in \( 2n\Delta - 3\Delta \) steps. The fastest possible coverage
of $G$ is via the traversal of a hamiltonian path in $G$ without waiting, which takes $n - 1$ steps, and $2\Delta(n - 1) > 2n\Delta - 3\Delta$. 

Here, $\mathcal{B}$ is starkly differentiated from $\mathcal{R}$ in that we have at least some ability to approximate in $\mathcal{B}$. See Chapter 5 for a further discussion of the relationship between these two classes.

### 4.3. $\mathcal{P}$: Periodic edges

In $\mathcal{P}$, similar to the case for $\mathcal{B}$, we can tighten the upper bound from Theorem 11:

**Corollary 4.** DMVP for a single agent in $\mathcal{P}$ is solvable in $O(pn^4 + n^22^n)$ time.

**Proof.** Since DMVP in $\mathcal{P}$ is bounded by $2pn$, no times later than this need to be considered. Thus, for $\mathcal{B}$, the running time of the algorithm in Theorem 11 reduces to $O(pn^4 + n^22^n)$. 

To exemplify the differences between $\mathcal{P}$ and $\mathcal{B}$, and motivate interest in the tractability of DMVP over $\mathcal{P}$, we first give the following simple example:

**Theorem 15.** For any $p$, there is a class of problems over combs, for which DMVP in $\mathcal{B}$ is NP-hard, but $\mathcal{P}$ is solvable by the simple online algorithm: take arms when you get to them.

**Proof.** Consider any TVG $\mathcal{G}$ in $\mathcal{P}$ whose underlying graph is a comb $G$ with backbone $B = b_0...b_{ak}$ and arms $A_0, ..., A_{\alpha}$, with each $A_i$ connected to $B$ at $b_{ik}$, for some $k, \alpha \in \mathbb{N}, k \geq 2p - 2$ (Figure 12). Suppose an agent $a$ starts at $b_0$. $a$ can either take $A_0$ immediately, or travel along $B$ to visit other arms and return to visit $A_0$ at a later time. Suppose the fastest an agent starting at $b_0$ can visit the leaf of $A_0$ and return to $b_0$ is $l$ steps. Then the longest this could possibly take
a starting at time 0 is \( l + (p - 1) \) steps, this worst case occurring when \( a \) must wait \( p - 1 \) steps for the fastest journey to become available. Suppose the fastest journey from \( b_0 \) to \( b_k \) takes \( k' \) steps. Then in the worst case, traveling from \( b_0 \) to \( b_k \) takes \( k' + (p - 1) \) steps. Suppose the fastest coverage of the remaining induced subgraph \( G' = G \setminus (A_0 \cup \{b_0, ..., b_{k-1}\}) \) takes \( m \) steps.

If \( G' \) has only one arm, the foremost journey from \( b_k \) to the leaf of this arm is clearly optimal. Assume that if \( G' \) has \( \alpha \) arms, then the following online algorithm results in an optimal solution: if at the endpoint of an unvisited arm, take that arm, otherwise visit the next unvisited vertex of \( B \). In the \( \alpha + 1 \) arm case, our agent starting at \( b_0 \) using this algorithm will complete coverage in no more than \( l + (p - 1) + k' + (p - 1) + m = l + k + (2p - 2) + m \) steps. But any solution in which \( a \) does not take \( A_0 \) first must cost at least \( l + k' + k + m \), as \( a \) must retraverse \( b_k ... b_0 \) on its way back to cover \( A_0 \). Since \( k \geq 2p - 2 \), the online solution must be minimal.

It is straightforward to modify Theorem 4 (i.e., by appropriately scaling up the underlying graph: elongating arms, extending the backbone, separating arms along the backbone, and adding an additional check tooth at \( b_0 \)) to show that DMVP over the above class of combs is NP-hard in \( \mathcal{B} \), for a single agent starting at \( b_0 \). \( \square \)
The quality of $\mathcal{P}$ we take advantage of above is that if the fastest journey between two nodes takes $d$ steps, the foremost journey can take no longer than $d + (p - 1)$, while in $\mathcal{B}$ it can be as bad as $d\Delta$. We again harness this effect in the following result, a stronger theorem in the context of our inapproximability results for $\mathcal{R}$ and $\mathcal{B}$ (Theorems 1 and 3):

**Theorem 16.** DMVP for a single agent in $\mathcal{P}$ over a spider is solvable in polynomial time, for fixed $p$.

**Proof.** Starting at the center $c$ of the spider, it is never useful for an agent to travel along any arm, unless it reaches a leaf. That is, an optimal solution is essentially an optimal visitation order of the leaves. We can set up a cost function $c(l, t)$ giving the minimal time it takes to travel from $c$ to leaf $l$ and back, starting at time $t$. Since $\mathcal{G}$ is periodic, $c(l, t) = c(l, t + kp) \forall k \in \mathbb{Z}$. Suppose the fastest journey from $c$ to $l$ and back has cost $m(l)$. Let extra time $e(l, t) = c(l, t) - m(l)$, be the cost above minimum incurred by traveling to $l$ and back starting at time $t$. $0 \leq e(l, t) < p \forall l, t$, since $a$ can always simply wait at most $p - 1$ steps for the periodically fastest journey to be available. For any $l$, there are only $p^2$ possible functions $e(l, t)$, since for all $0 \leq t \pmod{p} \leq p - 1$, $0 \leq e(l, t) < p$. Let $r(l, t)$ be the return time $\mod p$ of traveling to $l$ and back, that is, $c(l, t) = i \implies r(l, t) = t + i \pmod{p}$. Classify each $l$ by $e$ and $r$. Let $l_1 \equiv l_2 \iff e(l_1, t) = e(l_2, t)$ and $r(l_1, t) = r(l_2, t) \forall t$. Since for each $t$ there are $p$ choices for $e$ and $p$ choices for $r$, there are $p^3$ such equivalence classes. During a traversal of the spider, leaves with a common equivalence class are interchangeable, since at a given time, taking any of the same class will result in equivalent incurrence of cost above minimum as well as equivalent return time. Thus, a minimal traversal consists of traversing an ordering of arms corresponding to a sequence of equivalence classes $q_i$, such that the frequency of $q_i$ in the sequence is the number of arms classified as $q_i$. 
Notice that for any length $p$ traversed sequence $q_1, q_2, \ldots, q_p$, by the pigeonhole principle, there must be $q_i, q_j$ with $i < j$ such that $r(q_i, t_i) = r(q_j, t_j)$, where $t_i$ is the start time for traversing the $q_i$ arm, and $t_j$ is the start time for traversing the $q_j$ arm. Let $Q^t$ be a pattern if $Q$ is a sequence of equivalence classes with $0 < |Q| \leq p$, and starting at $c$ at time $t$, the traversal of $Q$ returns at some time $t' \equiv t \pmod{p}$. Furthermore, $Q^t$ is not a pattern if it contains any subpatterns, i.e., traversed subsequence with equivalent start and end time. Any length $p$ sequence must contain a pattern. An optimal solution can be decomposed into a sequence alternating between patterns and non-pattern subsequences between patterns. Any pattern can be removed from its location starting at time $t$ and inserted at any different location $t' \equiv t \pmod{p}$, since the fact that the pattern has the equivalent start and end time means adjacent journeys will be unaltered, due to the periodicity of $G$. In particular, any pattern $Q^t_1$ can be removed from its current location and inserted after any $Q^t_2$, without changing the cost of the solution. So, given any optimal solution, the following reordering process does not change the cost of the solution:

1. Divide the sequence into patterns $Q^t$ and stray arms in classes $q$ not in patterns
2. Set $i = 0$, $S = \{0\}$
3. Sequence all $Q^t_i$ together starting at $i$
4. Identify new patterns created by this move
5. Repeat 2 and 3 until nothing changes
6. Consider earliest start time $j$ of an arm such that $j \pmod{p} \notin S$ after final $Q_i$
7. Let $i = j$, and add $j$ to $S$.
8. Repeat 3 through 8 until nothing changes

After this process, all $Q_i$ are grouped together for all $i$, with fewer than $p$ stray arms separating each of these sequences of patterns, since in step 6, if there is no such $j$, then there must be fewer than $p$ arms left, otherwise there would be a pattern among these arms. Thus, the reordered sequence also ends with fewer than
$p$ stray arms. Since we started with an optimal solution, and the above process did not change the cost of the solution, there must be an optimal solution of this form. Since every such reordering begins with the $Q_0$ patterns, there are $(p-1)!$ orders in which the $p$ grouped sequences of patterns can show up. For each of these, there are at most $p$ clusters of stray arms each of length less than $p$. There are $O(p^5)$ ways to fill up these $O(p^2)$ slots with arms from the $p^3$ classes. Since there are $O((p^3)^p)$ possible patterns, there are $O(n(p^3)^p)$ ways to partition the remaining $O(n)$ arms in each class into patterns, and therefore $O(n(p^3)^p^p) = O(n(p^3)^{p+1})$ ways to partition all classes into patterns. This yields $O((p-1)!p^5n(p^3)^{p+1}) = O(n(p^3)^{p+1})$ possible solutions of the form reached by executing steps 1-8 above, at least one of which must be optimal. The cost of each of these possible solutions, of which there are polynomially many in $|\mathcal{G}|$, can be easily computed in time polynomial in $|\mathcal{G}|$. If $a$ does not start at $c$, but rather on some arm $A$, $a$ can either visit the leaf of $A$ before returning or $c$, or return to $c$ directly. Compute DMVP for the remaining arms in each of these two cases will yield an overall optimal solution still in time polynomial in $|\mathcal{G}|$. \hfill \Box

This polynomial runtime can be significantly improved, and extended to general trees, for the case of $p = 2$.

**Theorem 17.** DMVP for a single agent in $\mathcal{P}$ over a tree is solvable in $O(n)$ time, when $p = 2$.

**Proof.** Consider DMVP in $\mathcal{P}$, with $p = 2$, over a tree $T$, for an agent starting at root $o$ at time 0. The proof proceeds as follows: We first show by induction that there is always an optimal solution that never enters any of the subtrees of $o$’s children more than once. We then show that when covered in its entirety, each subtree is of one of three types: (fw11) fastest coverage with return to root is always available, (fw10) fastest coverage is available only at even times, and
(fw01) fastest coverage is available only at odd times. Alternating between fw10 and fw01 subtrees, and then taking the remaining subtrees in any order, before ending at a furthest leaf results in an optimal solution, as we maximize how many subtrees are traversed optimally. We can recursively compute the type and costs of covering the maximal subtree rooted at each node $v$, in $O(\deg(v))$ time for each.

Suppose $o$ has adjacent edges $e_1 = (o, u_1), ..., e_d = (o, u_d)$. Let $T_{u_i}$ be the maximal subtree rooted at $u_i$. Suppose agent $a$ starts at $o$. Recall that since $T$ is a tree, an optimal solution can be characterized as the set of leaves ordered by when they are visited.

Suppose an optimal solution ends on some leaf of $T_{u_i}$. Assume that when $T_{u_i}$ can be entered at most $k$ times during a solution, the solution that enters only once is still optimal. Suppose an optimal solution $S_{k+1}$ enters $T_{u_i} k + 1$ times. Then there is some non-empty subgraph $T_{k+1}^{u_i}$ of $T_{u_i}$ that $a$ covers the upon the $k$th entry, and $T_{k+1}^{u_i}$ it covers the upon the $(k + 1)$st entry. Let $T'$ be the subgraph of $T$ covered between the $k$th and $(k + 1)$st entries into $T_{u_i}$. When $a$ arrives at $o$ before entering $T_{u_i}$ for the $k$th time, consider an alternate completion resulting in an alternate solution $S_k$, in which $a$ instead immediately covers $T'$ and ends up back at $o$, at a cost of at most 1 plus the cost of this coverage in $S_{k+1}$ (which must occur in $S_{k+1}$ after the $k$th entry into $T_{u_i}$). If $S_k$ does incur a cost over $S_{k+1}$ for this traversal, then both $S_k$ and $S_{k+1}$ reach $o$ at times $\tau$ and $\tau'$ with equal parity. In $S_k$, $a$ completes coverage of $T$ during its $k$th entry to $T_{u_i}$, by covering $T_{k+1}^{u_i}$ and returning to $v_i$, then immediately covering $T_{k+1}^{u_i}$. If $\tau$ and $\tau'$ have equal parity, then $T_{k+1}^{u_i}$ is traversed in both $S_k$ and $S_{k+1}$ in equal time. Otherwise, $S_k$ may incur a cost of 1 over $S_{k+1}$ for this traversal. Therefore, the combined coverage of $T'$ and $T_{k+1}^{u_i}$ costs at most 1 more in $S_k$ than in $S_{k+1}$. $S_k$ may incur an additional cost of 1 over $S_{k+1}$ for its completing coverage of $T_{k+1}^{u_i}$, for a total incurrence of at most 2 over $S_{k+1}$ for these traversals. However, $S_{k+1}$ contains two additional traversals
of $e_i$, so the total cost of $S_{k+1}$ can be no better than that of $S_k$. Therefore, the solution that enters $T^u$ only once is optimal.

Recall (from the proof of Theorem 6) that in $P$ with $p = 2$, there are only three relevant dynamic edge types: 01, which are available at odd times but not even; 10, which are available at even times but not odd; and 11, which are available at all times. Let $T^u$ be a maximal subtree of $T$ rooted at $u$. Let $c_w(T^u, i)$ be the cost of the foremost journey covering $T^u$, starting at time $t \equiv i \mod 2$, with returning to end back at $u$. Let $c(T^u, i)$ be the cost of the foremost journey covering without necessarily returning to $u$. Let $m_w(T^u)$ be the fastest cost of covering $T^u$ with returning to end back at $u$, and $m(T^u)$ be the fastest cost of covering $T^u$ without necessarily returning to $u$. Notice that, since $p = 2$, the foremost cost of these coverings at any time can be no more than $c_w(T^u, i) = m_w(T^u) + 1$ and $c(T^u, i) = m(T^u) + 1$, respectively. Classify $T^u$ as $fw_{11}$ if a fastest coverage with return is always available; $fw_{10}$ if it is available at even times, but not odd; $fw_{01}$ if at odd times, but not even. Since each maximal subtree is covered independently, we can (as in the proof of Theorem 16) look for patterns in a solution given as an optimal ordering of child subtrees. Notice that in this case are only three patterns: $(fw_{11})$, $(fw_{10}, fw_{01})$, and $(fw_{01}, fw_{10})$. Given the above information for all of a tree $T^v$’s maximal child subtrees, we compute these values for $T^v$.

Following the characterization of an optimal solution given in the proof of Theorem 16, we can construct $S_0$, an optimal solution with return that starts at time $t \equiv 0 \mod 2$ by first taking as many copies of $(fw_{10}, fw_{01})$ as possible, since each of these will result in fastest traversals of the covered subtrees, taking one more $fw_{10}$ if possible, before taking the remaining subtrees in any order. We construct $S_1$, a similar solution with return that starts at time $t \equiv 1 \mod 2$, by taking an $fw_{01}$ subtree before constructing a time 0 solution in the same way as $S_0$. Since we know the form of an optimal solution, we can easily compute
the cost of each of these two solutions in $O(\deg(v))$. Then, for each $i \in \{0, 1\}$ and each subtree $T_u$ of $T_v$, we calculate the cost of covering $T_v$ without return starting at time $t \equiv i \mod 2$, such that $T_u$ is the final subtree covered. We can do this by subtracting the cost of covering $T_u$ in $S_i$, adding 1 if the removal of $T_u$ from $S_i$ necessarily decreases the number of subtrees taken optimally in $S_i$, and adding the cost of taking $T_u$ without return at the end of this new solution. The foremost cost of covering $T_v$ is the cost of the minimum solution over all $T_u$. Given the classification of $T_u$, this computation takes constant time for each $T_u$, and thus $O(\deg(v))$ overall. Given these costs, it is trivial to classify $T_v$. So, we can recursively compute all required values, at a cost of $O(\deg(v))$ per node, and thus $O(n)$ overall. The optimal cost of a complete solution is then $c(T, 0)$. □

We hypothesize that more efficient algorithms, such as the one for the $p = 2$ case, exist for this type of problem for greater values of $p$, and even general $p$, via this method of piecing together fast patterns. We have similar high hopes for larger classes of underlying graphs.
Open Problems and Discussion

This thesis has presented significant advances towards isolating the maximal class of graphs over which DMVP in $\mathcal{R}$ is solvable in polynomial time. We conjecture that this maximal class is the class of all graphs with polynomially many spanning trees, all of which have $O(\lg n)$ leaves. Furthermore, we conjecture that this class is equivalent for $\mathcal{R}$ and $\mathcal{B}$. But we are very interested in expanding this class with respect to $\mathcal{P}$, motivated by our solvability results for $\mathcal{P}$ over subclasses of trees. We have shown that for the case of $p = 2$, DMVP for a single agent over general trees can be computed in linear time. This result relies on the fact that we know how to optimally piece together patterns with period 2. New methods for finding optimal pattern sequences could greatly reduce computation for cases of $p > 2$. We are hopeful that DMVP in $\mathcal{P}$ will be shown to be poly-time solvable over arbitrary trees or at least bounded degree trees, for greater values of $p$ both fixed and not fixed, as well as over other larger classes of graphs. We have presented efficient algorithms for DMVP for $k$ agents in $\mathcal{R}$ over some restricted topologies, and we expect that there are even larger classes of underlying graphs for which DMVP for $k$ agents in $\mathcal{P}$ is tractable.

Considering $\mathcal{B}$ and $\mathcal{R}$, $\mathcal{B}$ is clearly differentiated from $\mathcal{R}$ in that we have at least some ability to approximate in $\mathcal{B}$. There remains, however, an important open question: Is there any class $\mathcal{G}$ of underlying graphs such that DMVP is NP-hard over $\mathcal{G}$ in $\mathcal{R}$, but not in $\mathcal{B}$? We are particularly interested in whether or not DMVP in $\mathcal{B}$ is NP-hard when the underlying graph is a star and $\Delta$ is fixed, in particular, when $\Delta = 2$. Note: The proof of Theorem 1 implies it is hard when $\Delta
is some relatively small function of the input. We conjecture that even for $\Delta = 2$
this problem is NP-hard, but the highly-restricted nature of the input makes an
answer to this problem more elusive than some of the others we have results for.
Towards an answer to this question, we give the following:

**Observation 2.** $\text{DMVP}(G, \{s\})$ for a single agent in $\mathcal{R}$ over a spider with
arms of uniform length $l$, e.g., a star (when $l = 1$), can be decided in polynomial
time, when $t$ disallows waiting, i.e., $t = 2n - l - d$, where $d$ is the topological
distance from $s$ to $c$.

**Proof.** Suppose $G$ is a spider with arms of uniform length $l$. Then, $G$ has
$n/l$ arms. Suppose $a$ starts at some vertex $s$ distance $d$ from the central vertex
$c$. If $t = 2n - l - d$, then the only solution can be a waiting-free spanning tree
traversal of $G$ starting at $s$. If $d > 0$, i.e., $s \neq c$, then the first leaf visited
must be the leaf of $s$’s arm. If $a$ starts at $c$, any arm can be traversed first. In
either case, starting at the first time $\tau$ that $a$ finds itself at $c$, $a$ must traverse the
$O(n/l) = \alpha$ remaining arms $a_1, \ldots, a_\alpha$ each in time $2l$, except for the final traversed
arm, whose leaf is reached in $l$ steps from $c$, completing the solution. Starting
at $\tau$, break the remaining time into $\alpha - 1$ length $2l$ time blocks $b_1, \ldots, b_{\alpha-1}$, and
a final length $l$ time block $b_\alpha$. For each $a_i$, for each $b_j$, we can straightforwardly
compute, in $O(l)$ time, whether or not $a$ can traverse $a_j$ and return to $c$ during $b_j$
without waiting. Deciding whether or not there exists a complete traversal of all
$\alpha$ arms without waiting then reduces to the problem of finding a perfect bipartite
matching between arms and time blocks, for which there are many known efficient
polynomial time algorithms, e.g., [30].

So, a proof of NP-hardness of $\text{DMVP}$ in $\mathcal{B}$ over stars must necessarily include
waiting in any optimal solution. Also, observe that $\text{weighted-DMVP}$ (namely,
when $\zeta(e, t)$ can be any positive integer) in $\mathcal{B}$ over stars is NP-hard. We get
this result from the proof of Theorem 3, simply by replacing each arm $A$ with a single edge with latency equal to the length of $A$. Clearly, we have a quite narrow area of fundamental importance between these cases. A solvability result in this area could have significant implications, as an assumption of time-boundedness could allow for computation of exact solutions, which may be crucial for long-scale problems such as those in planetary exploration.

Overall, our results show some instances where DMVP is tractable as well as showing that DMVP faces difficult computational challenges for some natural classes of underlying topologies and dynamics. These challenges motivate research into online, multi-agent solutions to the problem, since in many cases having a complete global view of the present and future does not appear to be very helpful; moreover, in agent-oriented applications ranging from software agents to mobile robots, the information available to teams of agents can be bounded both temporally and geographically, and such online, multi-agent approaches could be well suited to agent dynamics without diminishing tractability. We have begun to take steps in this direction using edge markovian TVG models [8]. In these types of stochastic environment, investigating interactive agent policies is an especially interesting direction to pursue.
Bibliography


[14] Casteigts, A., Flocchini, P., Quattrociocchi, W., Santoro, N.: Time-varying graphs and
dynamic networks. IJPED 27.5, 387–408 (2012)


Artificial Intelligence, 31, 113–126 (2001)


italiana (2007)

of ICRA, 727–734 (2005)


17–61 (1960)

[22] Hijazi, A., Nassar, N.: Using mobile agents for intrusion detection in wireless ad hoc

ACM SIGGRAPH, 46 (2005)


[25] Migas, N., Buchanan, W., McArtney, K.: Mobile agents for routing, topology discovery, and
automatic network reconfiguration in ad-hoc networks. Eng. of Comp.-Based Sys., 200–206
(2003)


39th STOC (2007)


