Multi-Robot Foremost Coverage of Time-Varying Graphs

Eric Aaron[1], Danny Krizanc[2(✉)], and Elliot Meyerson[2]

[1] Computer Science Department, Vassar College, Poughkeepsie, NY, USA
eaaron@cs.vassar.edu
[2] Department of Mathematics and Computer Science, Wesleyan University, Middletown, CT, USA
{dkrizanc,ekmeyerson}@wesleyan.edu

Abstract. In this paper we demonstrate the application of time-varying graphs (TVGs) for modeling and analyzing multi-robot foremost coverage in dynamic environments. In particular, we consider the multi-robot, multi-depot Dynamic Map Visitation Problem (DMVP), in which a team of robots must visit a collection of critical locations as quickly as possible, in an environment that may change rapidly and unpredictably during navigation. We analyze DMVP in the context of the $\mathcal{R} \supset \mathcal{B} \supset \mathcal{P}$ TVG hierarchy. We present exact offline algorithms for k robots on edge-recurrent TVGs ($\mathcal{R}$) over a range of topologies motivated by border coverage: an $O(Tn)$ algorithm on a path and an $O(T\frac{n^2}{k})$ algorithm on a cycle (where T is a time bound that is linear in the input size), as well as polynomial and fixed parameter tractable solutions for more general notions of border coverage. We also present algorithms for the case of two robots on a tree (and outline generalizations to k robots), including an $O(n^5)$ exact algorithm for the case of edge-periodic TVGs ($\mathcal{P}$) with period 2, and a tight poly-time approximation for time-bounded edge-recurrent TVGs ($\mathcal{B}$). Finally, we present a linear-time $\frac{12\Delta}{5}$-approximation for two robots on general graphs in $\mathcal{B}$ with edge-recurrence bound Δ.

1 Introduction

For mobile robot applications such as multi-robot surveillance, search and rescue, patrol, and inspection tasks, problems are often formulated as graph coverage problems. In many such applications, the robots may navigate in dynamic environments that can change unpredictably during navigation, but conventional static graph formulations do not represent those essential dynamics. We address this issue by adopting recent formulations of *time-varying graphs* (TVGs) to enable analysis of multi-robot team navigation in dynamic environments. In particular, in this paper we present results for the multi-robot, multi-depot *Dynamic Map Visitation Problem* (DMVP), in which a team of robots must visit a collection of critical locations on a map (graph) as quickly as possible, but the environment may change during navigation. We present efficient offline algorithms, including a fixed parameter tractable solution, for an arbitrary number of robots over a

J. Gao et al. (Eds.): ALGOSENSORS 2014, LNCS 8847, pp. 22–38, 2015.
DOI: 10.1007/978-3-662-46018-4_2

range of topologies motivated by border coverage (Sect. 2), and for two robots on a tree (Sect. 3); details of our main results are summarized in Sect. 1.2.

Many approaches to coverage problems [10,11] (including border coverage [14,23]) are based on static graph representations, as are related combinatorial optimization problems such as the k-traveling repairman problem, k-traveling salesman problem, etc. [4,15]. DMVP is distinct from these other problems, with crucial and distinguishing aspects of DMVP including (1) robots can start at any number of distinct depots, and (2) robots need not return to their depot after completion of coverage. Permitting multiple depots allows for the teaming of geographically disjoint robots; while completing a series of heterogeneous tasks, robots may not be together when a map visitation call is warranted. The absence of a requirement for return ensures that the singular goal is timely coverage completion, which is important for time-sensitive inspection tasks or other applications.

The most fundamental difference between DMVP and related problems is that DMVP employs a TVG representation of the environment, which can capture variation in graph structure over time in ways that static graphs cannot. A TVG [8] is a five-tuple $\mathcal{G} = (V, E, \mathcal{T}, \rho, \zeta)$, where $\mathcal{T} \subseteq \mathbb{T}$ is the *lifetime* of the system, *presence function* $\rho(e,t) = 1 \iff$ edge $e \in E$ is available at time $t \in \mathcal{T}$, and *latency function* $\zeta(e,t)$ gives the time it takes to cross e if starting at time t. The graph $G = (V, E)$ is called the *underlying graph* of $\mathcal{G}$, with $|V| = n$. As in [2,16,20], we consider the discrete case in which $\mathbb{T} = \mathbb{N}$, edges are undirected, and all edges have uniform travel cost 1. If agent a is at u, and edge (u, v) is available at time τ, then a can take (u, v) during this time step, visiting v at time $\tau + 1$. As a traverses $\mathcal{G}$ we say a both *visits* and *covers* the vertices in its traversal, and we will henceforth use these terms interchangeably. $\mathcal{J} = \{(e_1, t_1), ..., (e_k, t_k)\}$ is a *journey* $\iff$ $\{e_1, ..., e_k\}$ is a walk in G (called the *underlying walk* of $\mathcal{J}$), $\rho(e_i, t_i) = 1$ and $t_{i+1} \geq t_i + \zeta(e_i, t_i)$ for all $i < k$. The *topological length* of $\mathcal{J}$ is k, the number of edges traversed. The *temporal length* is the duration of the journey: (*arrival date*) − (*departure date*). Given a date t, a journey from u to v departing on or after t whose arrival time is soonest is called *foremost*; whose topological length is minimal is called *shortest*; and whose temporal length is minimal is called *fastest*.

In [8], a hierarchy of thirteen TVG classes is presented. In related work on exploration [2,16,19], broadcast [7], and offline computation of optimal journeys [6], focus is primarily on the chain $\mathcal{R} \supset \mathcal{B} \supset \mathcal{P}$, which enforce natural constraints for mobile robot applications: $\mathcal{R}$ (recurrence of edges) is the class of all TVG's $\mathcal{G}$ such that G is connected, and $\forall e \in E, \forall t \in \mathcal{T}, \exists t' > t$ s.t. $\rho(e, t') = 1$; $\mathcal{B}$ (time-bounded recurrence of edges) is the class of all TVG's $\mathcal{G}$ such that G is connected, and $\forall e \in E, \forall t \in \mathcal{T}, \exists t' \in [t, t + \Delta)$ s.t. $\rho(e, t') = 1$, for some Δ; $\mathcal{P}$ (periodic edges) is the class of all TVG's $\mathcal{G}$ such that G is connected, and $\forall e \in E, \forall t \in \mathcal{T}, \forall k \in \mathbb{N}, \rho(e, t) = \rho(e, t + kp)$ for some p, the *period* of $\mathcal{G}$.

We are interested in solving the following problem:

Problem. *Given a TVG $\mathcal{G}$ (in class $\mathcal{R}$, $\mathcal{B}$ or $\mathcal{P}$) and a set of starting locations S for k agents in $\mathcal{G}$, the TVG foremost coverage or Dynamic Map Visitation*

Problem (DMVP) is the task of finding journeys starting at time 0 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 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 arrival date later than t.

For the minimization version of the problem $DMVP(\mathcal{G}, S)$ and the corresponding decision problem $DMVP(\mathcal{G}, S, t)$, the input is viewed as a sequence of graphs G_i each represented as an $n \times n$ adjacency matrix, with an associated integer duration t_i, i.e., $\mathcal{G} = (G_1, t_1), (G_2, t_2), ..., (G_m, t_m)$, where G_1 appears initially at time zero (see [9,21,22] for alternative views of TVGs). Let $T = \sum_{i=1}^{m} t_i$. We know from [2] that we can run an $O(nm)$ preprocessing step that lets us presume that $T < 2\,\mathrm{nm}–3\,\mathrm{m}$, (that is, T is at worst linear in $\mathcal{G}$), and enables $O(1)$ edge presence lookups $\rho(e, \tau)$, without affecting asymptotic runtime of any of the algorithms presented below. We think of the input $\mathcal{G}$ as a temporal subgraph of some TVG $\mathcal{G}_\infty$ with lifetime $\mathbb{N}$ and the same edge constraints as $\mathcal{G}$. Thus, the limited information provided in $\mathcal{G}$ is used to find journeys (which may have temporal length greater than T) that cover G, for agents in $\mathcal{G}_\infty$.

1.1 Related Results

The problem most similar to (but distinct from) DMVP is the minmax k-traveling salesman problem [12,27], in which all robots start at and return to a single depot on a static graph. Approximation algorithms have been given that forgo the single depot requirement, but still require a return to multiple initial depots [4,26]. To the best of our knowledge, no previous work has addressed the case of exact algorithms for multiple agents either without return or with multiple depots, even for the static case. A pseudo-polynomial time algorithm for any constant $k > 1$ agents on a tree for the k-traveling salesman problem (single depot with return) is presented in [12]. A pseudo-polynomial solution for the weighted tree case is given in [27]. This algorithm runs in $O(n^3)$ for the restriction to two robots and unweighted edges (Lemma 1). We sequentially generalize this to DMVP by (1) allowing multiple depots, (2) not requiring robots to return to their depots, and (3) incorporating TVG models, namely, $\mathcal{P}$ and $\mathcal{B}$ (Sect. 3).

Heuristics for boundary coverage for multiple robots are considered in [14], in which the problem is reduced to k-rural postman over a static graph extracted from a continuous environment. This graph extraction procedure motivates our result on "border coverage" graphs in $\mathcal{R}$ (Theorem 3).

The complexity of DMVP for a single agent was explored in [2], in which it was shown that in the edge-recurrent TVG class $\mathcal{R}$ it is NP-hard to approximate within any factor, even over max-degree 3 trees, and stars (i.e., trees with at most one vertex of degree greater than 1). (A related result was derived independently in [25].) The periodic case $\mathcal{P}$, even with period $p = 2$, was shown to be NP-hard over a larger class of graphs than the static case MVP, which is hard even over

trees when k is part of the input [1]. Other hardness results for problems over TVGs have been shown for computing strongly connected components [5] and dynamic diameter [18].

1.2 Main Results

We present algorithms for DMVP for k agents in $\mathcal{R}$ over a range of topologies motivated by border coverage: an $O(Tn)$ algorithm to optimally solve DMVP on a path, an $O(T\frac{n^2}{k})$ algorithm on a cycle, a polynomial solution for the border graph of a planar region divided into a constant number of components, and a fixed parameter tractable solution for any m-leaf c-almost tree, for parameters m and k, and constant c. We demonstrate a fundamental hardness separation between $\mathcal{P}$ and static graphs for all fixed k. We then consider the case of trees in $\mathcal{P}$ with $p = 2$ and give a $O(n^5)$ algorithm for the case of two agents. We also give an $O(n^3)$ algorithm for tight approximation for two agents on a tree in $\mathcal{B}$. Finally, we present a linear-time $\frac{12\Delta}{5}$ approximation for two agents on general graphs in $\mathcal{B}$ with edge-recurrence bound Δ. Corresponding generalizations to k agents are outlined here and will appear in the full version of this paper.

2 k-Agent Border Coverage in $\mathcal{R}$

DMVP on paths, cycles and more general classes of graphs is motivated by border coverage, e.g., for security. Coverage of a path corresponds to securing critical points along the border between any two adjacent connected planar regions, neither of which surrounds the other, while coverage of a cycle corresponds to securing the complete border of any simply connected planar region.

Theorem 1. *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. Orient P left-to-right, with v_1 the leftmost vertex. Without loss of generality, suppose $s_1, ..., s_k$ are ordered from left to right. Note that if two or more agents start at the same vertex s_i, simply sending two of them in opposite directions will be trivially optimal, thereby reducing the problem to two instances of DMVP over edge-disjoint subpaths $v_1...s_i$ and $s_i...v_n$, which can be solved independently.

Assume no two agents start at the same node. The idea is to compute for each vertex $u \in s_1...v_n$ the optimal cost of the solution to the DMVP subproblem over $v_1...u$ for all agents starting on or to the left of u. Call this cost $c(u)$. We can compute all $c(u)$ from left to right, and finally get the result $c(v_n)$ for DMVP for all k agents over P (Algorithm 1). 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

Algorithm 1. DMVP-Path($\mathcal{G}, \{s_1, ..., s_k\}$)

for all $v \in s_1 ... v_n$ **do** ▷ Initialize c
  $c(v) = \infty$
for $i = 1, ..., k$ **do**
  $lBoundary = s_i$ ▷ Evaluate all left-first journeys for a_i
  if $i = 1$ **then**
    $lBoundary = v_1$
  while $lBoundary \notin \{s_{i-1}, \emptyset\}$ **do** ▷ Try every possible left endpoint
    $t = 0$
    $loc = s_i$
    $turned = eval = False$ ▷ evaluate solution?
    while $loc \notin \{\emptyset, s_{i+1}\}$ and $t < T$ **do** ▷ enter at most T times
      if $loc = lBoundary$ **then**
        $turned = True$
      if $turned = True$ and $loc = s_i$ **then**
        $eval = True$
      if $eval = True$ **then**
        $c(loc) = \min(c(loc), \max(c(lBoundary.lNode), t))$
      if not $turned$ and $\rho(loc.lEdge, t) = 1$ **then**
        $loc = loc.lNode$
      if $turned$ and $\rho(loc.rEdge, t) = 1$ **then**
        $loc = loc.rNode$
      $t = t + 1$
    $lBoundary = lBoundary.lNode$
  $rBoundary = s_i$ ▷ Evaluate all right-first journeys for a_i
  if $i = k$ **then**
    $rBoundary = v_n$
  while $rBoundary \notin \{s_{i+1}, \emptyset\}$ **do** ▷ Try every possible right endpoint
    $t = 0$
    $loc = s_i$
    $turned = eval = False$ ▷ evaluate solution?
    while $loc \notin \{s_{i-1}, \emptyset\}$ and $t < T$ **do** ▷ enter at most T times
      if $loc = rBoundary$ **then**
        $turned = True$
      if $turned = True$ and $loc = s_i$ **then**
        $eval = True$
      if $eval = True$ **then**
        $c(rBoundary) = \min(c(rBoundary), \max(c(loc.lNode), t))$
      if not $turned$ and $\rho(loc.rEdge, t) = 1$ **then**
        $loc = loc.rNode$
      if $turned$ and $\rho(loc.lEdge, t) = 1$ **then**
        $loc = loc.lNode$
      $t = t + 1$
    $rBoundary = rBoundary.rNode$
  return $c(v_n)$

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 $O(T|R_1|) + O(T|L_2|) + O(T|R_2|) + ... + O(T|L_{k-1}|) + O(T|R_{k-1}|) + O(T|L_k|) = O(Tn)$. □

Theorem 2. *DMVP for k agents in $\mathcal{R}$ on a cycle is solvable in $O(T\frac{n^2}{k})$ time.*

Proof. Consider DMVP over the cycle $C = v_0v_1...v_nv_0$ for k agents $a_1, ..., a_k$ ordered clockwise around the cycle at locations $s_1, ..., s_k$, respectively. If any two agents start at the same node, then sending them in opposite directions will be optimal, thereby reducing the problem to DMVP on a path, which can be solved with Algorithm 1 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, $d \leq \lfloor \frac{n}{k} \rfloor$. 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 1 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 $O(T\frac{n^2}{k})$ runtime. □

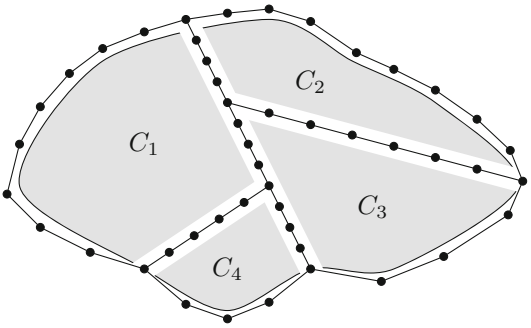

Fig. 1. Border coverage graph extracted from a planar region (gray) subdivided into four components.

The next result corresponds to a more general notion of border coverage akin to that addressed in [14]. Consider any simply connected planar region divided by throughways into some number of subregions, e.g., a complex of secure buildings or zones. The *border coverage graph* of such a subdivided region is the graph induced by the coverage of critical points along the union of the subregions' borders, e.g., Fig. 1.

Theorem 3. *DMVP for k agents in $\mathcal{R}$ is solvable in $O(Tn^{6c+1})$ time, when G is the border coverage graph of a simply connected planar region divided into c subregions, for c constant.*

Proof. Suppose R is a simply connected planar region divided into c subregions, for c constant. Call a path $P \subset G$ a *through-path* if the endpoints of P have degree greater than two and all intermediate vertices of P have degree two. Let c_1 be the number of through-paths in G. c_1 can be bounded by considering how R is subdivided. Let G_i be the border coverage graph corresponding to R divided into $i < c$ subregions. We create a new subregion by adding a through-path P between two vertices of G_i, such that all vertices of P are internal to R and no edge of P crosses an edge of G_i. This addition creates another through-path for each endpoint of P that had degree two before the addition. Thus, at most three new through-paths are added for each subregion of R, i.e., $c_1 < 3c$.

In an optimal solution, the agents that start on a through-path $P = u...v$ but never reach u or v must together cover a set of vertices whose induced graph is a subpath of P. For each P, there are $O(n^2)$ such subpaths. It would never be better for an outside agent to enter P in order to cover a vertex between two disjoint subpaths, as it must cross over at least one agent that never leaves P, and the remainder of their journeys could be swapped at no cost. So we forbid outside agents to travel along these subpaths. From Theorem 1, DMVP for each of these subpaths can be computed in $O(Tn)$.

Selecting the subpath these agents cover for every through-path induces a subset of the remaining vertices that must be covered to complete coverage, namely, the vertices adjacent to but not included in any subpath. There

are $O(n^{2c_1})$ ways to make this selection. If the internally-covered subpath of a through-path P with endpoints u and v is empty, then it must be that no agents started between u and v, so in addition to the $O(n)$ choices for pairs of vertices adjacent to a subpath of P of length 0, there are two further ways for outside agents to complete coverage of P: by at some point traveling directly from u to v, or from v to u, covering all of P along the way. At most two outside agents are required to cover the remainder of each path, so in an optimal solution at most $2c_1$ agents leave their start paths. For any P, the agents that could leave are each of the at most $2c_1$ closest to u and v, resp. There are $c_2 = (2c_1)^{4c_1^2}$ ways to partition the remaining elements to cover between all agents that could leave their start paths, and after running an $O(Tn^3)$ all-pairs-all-times-foremost-journey preprocessing step [2], DMVP for each agent can be computed in $O(4c_1^2 2^{2c_1})$. Running this for each agent for the $O(n^{2c_1})$ ways to cover all paths and computing internal path costs yields a total runtime of $O(n^{2c_1})(O(4c_2c_1^2 2^{2c_1}) + O(c_1Tn)) + O(Tn^3) = O(Tn^{6c+1})$. □

Pointing towards further generalizations, the following theorem extending Thm. 10 in [2] applies to a slightly larger classes of graphs and includes the number of agents as a parameter in an fixed parameter tractable (FPT) solution. We will give the complete proof in the full version.

Theorem 4. *DMVP for k agents in $\mathcal{R}$ is fixed parameter tractable, when G is an m-leaf c-almost-tree, for parameters m and k, and c constant.*

Proof sketch. For all $t < T$, consider the decision variant over m-leaf trees. Partitioning the leaf set among agents (k^m ways) and using the single agent $O(Tn^{3+c}f(m))$ algorithm [2] for each guarantees coverage of everything but the union of shortest paths between depots. If an edge in such a path has not already been covered this creates a cut with agents confined to subtrees. There are 2^k ways to select which paths are cut. Such a selection induces a tree of subproblems which can be solved in a bottom-up fashion, fixing cut points along the way. There will always be an unsolved subproblem of degree no more than one. Fix the cut point for this subproblem as far as possible given the time bound t by testing the instance corresponding to each of the $O(n)$ possible cut points along the path. This factor of n can be pulled out to keep the algorithm FPT for k, m. It is straightforward to extend this idea to c-almost-trees. □

3 Two Agents on a Tree

We know from [2] that DMVP for k agents on a tree is hard in $\mathcal{B}$, regardless of k, even over spiders (i.e., trees in which at most one vertex has degree greater than 2). However, for a single agent in $\mathcal{P}$, DMVP can be solved in polynomial time over spiders for fixed p, and in linear time on arbitrary trees when $p = 2$. Since in $\mathcal{P}$ we are able to efficiently solve DMVP over a wider range of graph classes than in $\mathcal{B}$ or $\mathcal{R}$ [2], to show that for multiple agents $\mathcal{P}$ is fundamentally more complex than the class of static graphs, we demonstrate that for DMVP

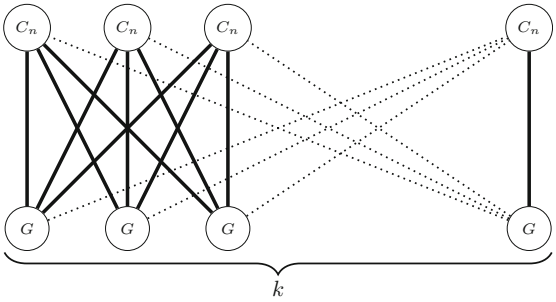

Fig. 2. Graph class for which DMVP is NP-hard in $\mathcal{P}$ with $p = 2$, but trivially in P when $p = 1$. Each thick edge represents the edges in the complete biparitite graph linking vertices in a C_n to vertices in a G.

for k agents there is a hardness separation between $\mathcal{P}$ with $p = 2$, and $p = 1$, for all k. Note that when $p = 2$, edges can only be one of three possible dynamic types: (01) available only at odd times, (10) available only at even times, (11) available at all times.

Theorem 5. *For all $k \geq 1$, there is a class of graphs C such that DMVP in $\mathcal{P}$ for k agents over graphs in C is trivial when $p = 1$, but NP-hard when $p = 2$.*

Proof. For any graph G with an even number of vertices $v_0, ..., v_{n-1}$, take k copies of G and k copies of $C_n = c_0...c_{n-1}c_0$, a cycle of length n. Add edges to form a complete bipartite graph linking vertices in each C_n to each G (see Fig. 2). For $p = 2$, let all original edges of G be of type 11. Let all (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}) and (c_i, c_{i+1}) be of type 10 when i is even and 01 when i is odd, where indices are taken mod n. Suppose each agent a_i starts at a distinct v_0. If $t = 2n - 1$, a_i must completely cover G before moving to a C_n, to avoid waiting at a vertex of C_n for a time step on the way back to a G, and thus effectively solve HAM-PATH [17] on G. However, when $p = 1$, each agent simply jumps repeatedly from a G to a C_n, since every uncovered vertex across the bipartite cut is always available. □

Now, even DMVP restricted to static graphs (also known as MVP) is in general NP-hard on trees for k agents, but for a single agent it can be solved in linear time [1]. What about DMVP when $k = 2$? We build up to an exact polynomial solution for DMVP on a tree for two agents a_1, a_2 in $\mathcal{P}$ for the $p = 2$ case, and a tight approximation in $\mathcal{B}$ for all Δ, via a series of related lemmas partially-ordered by constraints, see Fig. 3. The base result (Lemma 1) is implied by an upper bound established in [27], but the further results are, to our knowledge, novel generalizations, with our main result being an $O(n^5)$ solution for DMVP in $\mathcal{P}$ for two agents with $p = 2$ (Theorem 7).

Lemma 1. *MVP with return for two agents starting at a single depot on a tree can be solved in $O(n^3)$ time.*

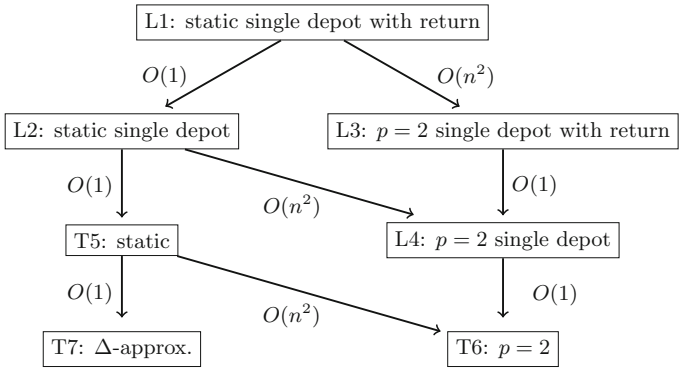

Fig. 3. Poset of results leading to solutions for two-agent DMVP on a tree; arrows indicate increasing factors of complexity as constraints are loosened.

Proof sketch. This result is implied as a special case in [27]. (An $O(n^6)$ algorithm is given in [12].) We give the following proof idea:

At each node v (whose maximal subtree is denoted G^v) from the leaves up to the starting depot, i.e., root s, we compute and store possible pairs of costs for a_1 and a_2 covering and returning to the root of the maximal subtree rooted at v, by iterating over the pairs of costs $(c_1, c_2)_{u_i}$ associated with covering each of v's children $u_1, ..., u_{\deg v}$, to compute partial solution costs $(c_1, c_2)^i_v$, corresponding to possible pairs of costs covering the subtrees $G^{u_1}, ..., G^{u_i}$. However, each new cost pair $(c_1, c_2)^i_v$ is only stored if c_2 is less than the current best cost associated with c_1. Each cost is bounded by $2n - 3$, and each child is iterated over only once, taking $O(n^2)$ to combine its costs $(c_1, c_2)_{u_i}$ with the costs $(c_1, c_2)^{i-1}_v$ of covering the previous branches of the subtree to get all $(c_1, c_2)^i_v$, yielding the $O(n^3)$ total runtime. □

The following extension drops the constraint of returning to root.

Lemma 2. *MVP for two agents starting at a single depot on a tree can be solved in $O(n^3)$ time.*

Proof. Follow the same method describe in the proof of Lemma 1, except now at each node v store pairs of costs for covering G^v for each of the following four cases: both a_1 and a_2 return to v ($(c^r_1, c^r_2)_v$), a_1 returns to v but not a_2 ($(c^r_1, c_2)_v$), a_2 returns to v but not a_1 ($(c_1, c^r_2)_v$), neither returns to v ($(c_1, c_2)_v$). Partial solutions are then updated for each return type: $(c^r_1, c^r_2)^{i-1}_v$ combined with $(c^r_1, c^r_2)_{u_i}$, $(c^r_1, c_2)_{u_i}$, $(c_1, c^r_2)_{u_i}$, and $(c_1, c_2)_{u_i}$ to get $(c^r_1, c^r_2)^i_v$, $(c^r_1, c_2)^i_v$, $(c_1, c^r_2)^i_v$, and $(c_1, c_2)^i_v$, resp.; $(c^r_1, c_2)^{i-1}_v$ combined with $(c^r_1, c^r_2)_{u_i}$ and $(c_1, c^r_2)_{u_i}$ to get $(c^r_1, c_2)^i_v$ and $(c_1, c_2)^i_v$; $(c_1, c^r_2)^{i-1}_v$ combined with $(c^r_1, c^r_2)_{u_i}$ and $(c^r_1, c_2)_{u_i}$ to get $(c_1, c^r_2)^i_v$ and $(c_1, c_2)^i_v$; $(c_1, c_2)^{i-1}_v$ combined with $(c^r_1, c^r_2)_{u_i}$ to get $(c_1, c_2)^i_v$. That is, with-return costs are added to with-return costs to get new with-return costs, as the journey must end on some later branch; without-return costs are added to with-return costs to get new without-return costs that end on the current branch; with-return

costs are added to without-return costs to get new without-return costs that end up on some previous branch. Updating cost pairs for all four return types incurs only a constant factor runtime increase over the return to root case. □

This is generalized now to the case of multiple depots; the standard MVP formulation.

Theorem 6. *MVP for two agents on a tree can be solved in $O(n^3)$ time.*

Proof. Suppose a_1 starts at s_1 and a_2 starts at s_2. Let $P = (s_1 = p_1)p_2...p_{l-1}$ $(p_l = s_2)$ be the unique simple path from s_1 to s_2.

First, note that if a_1 and a_2 do not cross paths, that is, there is no $v \in P$ such that both a_1 and a_2 include v in their journeys, then the subtrees covered by a_1 and a_2 will be disjoint, reducing the problem to two instances of MVP on a tree for a single agent, each of which can be solved independently in $O(n)$ [1]. There are $O(n)$ ways to cut P so that the journeys are disjoint, so trying each of these possibilities takes $O(n^2)$, which is subsumed by the cost of considering non-disjoint solutions.

Assuming the optimal journeys are not disjoint, using the algorithm described in Lemma 2, run for all $v \in P$ MVP for two agents starting at a single depot for the maximal subtree rooted at v that is edge-disjoint from P, generating all resulting potential cost pairs. Now, we build up solutions from left-to-right, i.e., from s_1 to s_2. After considering each p_i along P, we want all cost pairs for all four cost pair cases (both return, only a_1 returns, etc.) of covering all of G excluding the branches rooted at all p_j, for all $j > i$. With-return costs are added to with-return costs to get new with-return costs; without-return costs are added to with-return costs to get new without-return costs that end on p_i's branch; with-return costs are added to without-return costs to get new without-return costs that end up on some previous branch rooted at p_k, for some $k < i$. Additional cost for traversing P is accumulated along the way: each time a_1 precedes to the next vertex of P, 1 is added to the cost of a_1's with-return costs (2 to without-return); $2|l - i|$ added to a_2's costs when p_i is selected to be its furthest vertex reached, and $|l - j|$ subtracted when p_j's branch is marked as the final branch a_2 enters, i.e., when p_j's without-return costs are added to previous with-return costs for $p_1...p_{j-1}$. Updating costs at each branch again takes $O(n^3)$, so the cost of the overall solution remains $O(n^3)$. □

That concludes our results for the static case. We now generalize these results to the case of TVGs in $\mathcal{P}$.

Lemma 3. *DMVP with return for two agents starting at a single depot on a tree in $\mathcal{P}$ can be solved in $O(n^5)$, when $p = 2$.*

Proof. This case runs similar to Lemma 1, but since we are in $\mathcal{P}$, we must be careful about how we build up solutions, as it matters in which order branches are taken. From [2], we know that with $p = 2$, each agent can enter each branch at most once, and that each branch can in an $O(n)$ pre-processing step be marked as either 01, fastest journey available only at even times; 10, fastest journey only

available at odd times; or 11, fastest journey always available. Note that this also applies to the subbranch covered by each agent. Furthermore, the optimal way for a single agent to cover any set of classified branches is to alternate between taking 01's and 10's as many times as possible, before taking the remaining branches in any order. So, given a start time along with the difference d_i between the number of 01's and 10's in a partial solution for a_i's coverage of G^v, we can add a partition of a new branch and check in constant time exactly how the cost of our solution will be affected.

Computing from the leaves up, as in Lemma 1, we store all possible pairs of costs of covering the maximal subtree rooted at v, but now we store separate pairs of costs for four cases defined by whether each a_i reaches v at an odd or even time. This adds only a constant factor overhead, and given a pair of costs and start times, we can in constant time compute whether the type τ_i of each of the two journeys is 01, 10, or 11. Storing all possible d_i for each cost pair means $O(\deg(v)^2 n)$ tuples (c_1, c_2, d_1, d_2) are stored at each branch, with branch updates taking $O(\deg(v)^2 n^2)$, as $(c_1, c_2, d_1, d_2)_v^{i-1}$ partial solutions are combined with $(c_1, c_2, \tau_1, \tau_2)_{u_i}$, yielding a cost of $O(\deg(v)^3 n^2)$ per node, and hence $O(n^5)$ overall. □

Lemma 4. *DMVP for two agents starting at a single depot on a tree in $\mathcal{P}$ can be solved in $O(n^5)$ time, when $p = 2$.*

Proof. In a similar manner to the extension from Lemma 1 to Lemma 2, we now store cost pairs for each of the four return cases for a_1 and a_2. Branch types can still be maintained as in Lemma 3 in order to preserve optimal orderings, so the algorithm again runs in $O(n^5)$. □

Theorem 7. *DMVP for 2 agents on a tree in $\mathcal{P}$ can be solved in $O(n^5)$ time, when $p = 2$.*

Proof. Again, let $P = (s_1 = p_1)p_2 ... p_{l-1}(p_l = s_2)$ be the unique simple path from s_1 to s_2, and assume a_1's and a_2's journeys are not disjoint, since we can solve each of the $O(n)$ disjoint instances in $O(n)$. We adopt a similar though more involved version of the left-to-right dynamic programming approach from the proof of Lemma 6, as the order now matters in which each agent takes its portion of each subtree b_i (i.e., subtrees rooted at each p_i but disjoint from P). First, compute all pairs of costs for covering each b_i in $O(n^5)$ via Lemma 4.

Suppose the final subtree taken by a_1 in an optimal solution is b_j. Then, a_1 can only take its assigned sections of $b_1, ..., b_{j-1}$ as it moves towards s_2 for the first time, since each agent can enter any subtree at most once in $\mathcal{P}$ with $p = 2$. Suppose the closest a_1 gets to s_2 is p_k. Then all of $b_{j+1}, ..., b_{k-1}$ can be taken either on the way from p_j to p_k, or on the way back. A similar case applies for a_2. So, as we consider each branch from s_1 to s_2, we build up partial solution costs for a_1 and a_2 in two directions at once: outside-in for a_1, and inside-out for a_2. That is, suppose that through $i - 1$ branches we have stored all $(c_1(\rightarrow), c_1(\leftarrow), t)_P^{i-1}$, where $c_1(\rightarrow)$ is the cost of the forward journey so far, $c_1(\leftarrow)$ is the cost of the reverse journey (i.e., after covering its portion of b_k),

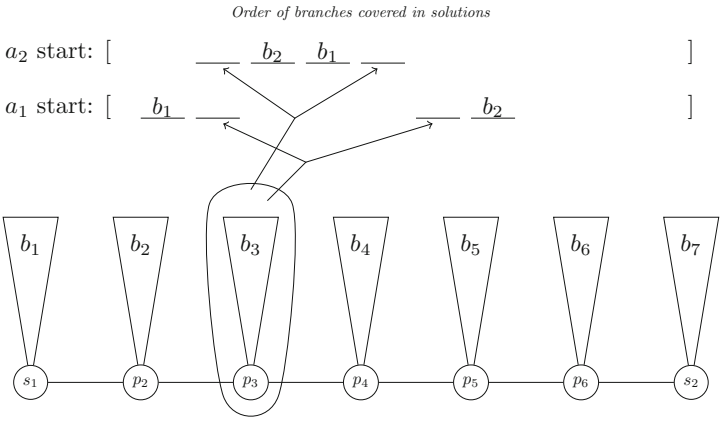

Fig. 4. Possible ways to update costs for a pair of partial solutions to include each agent's coverage of b_3, assuming a_1 ends on b_2, a_2 ends on some branch b_i, with $i > 3$, and a_2 takes b_2 on the way to b_1. In this case, both a_1 and a_2 can take their portion of b_3 either directly after b_1 or directly preceding b_2.

and t is the start time for the reverse journey; and $(c_2, t)_P^{i-1}$, where c_2 is the costs of a_2 covering its portions of $b_1, ..., b_{i-1}$, assuming p_{i-1} is reached by a_2 for the first time at time t. Update partial solutions to include b_i in the following way: for a_1, if a branch has been taken without return, further branches can be taken either directly after all forward journeys or directly before all reverse journeys, otherwise, all branches can only be taken on forward journeys, except of course for the branch taken without return, which must be taken last; for a_2, if a branch has been taken without return, further branches must be taken with return directly preceding existing solutions, otherwise, further branches can either be taken directly before existing solutions *or* directly after (see Fig. 4), and all must be taken with return. The cost of the final branch taken by a_1 is succinctly inserted between the forward and backward costs. Additional costs accumulated via the traversal of edges of P are added in as in Theorem 6, but now taking into account the edge type (i.e., 01,10, or 11), and the time parity at which the edge is reached. Running these updates for each time parity, each return case, and each location of the branch in an optimal ordering incurs together only constant factor overhead. Storing both $(c_2, t)_P^i$ for all $(c_1(\rightarrow), c_1(\leftarrow), t)_P^i$, takes $O(n^2)$ space, but we can reduce this to $O(n)$ by compactly representing the solution cost by the sum of $c_1(\rightarrow)$ and $c_1(\leftarrow)$, and a bit for storing the parity of each. The update at each branch still takes $O(n^2)$ to compute all possible cost pair cases for the new partial solutions, so the full iteration from p_1 to p_l takes $O(n^3)$, and the initial $O(n^5)$ runtime dominates. □

We can also apply Theorem 6 to get a tight approximation for two agents on a tree in $\mathcal{B}$:

Theorem 8. *DMVP for two agents on a tree in $\mathcal{B}$ can be Δ-approximated in $O(n^3)$ time $\forall\ \Delta > 1$. This approximation is tight.*

Proof. The cost to cover a tree G in $\mathcal{B}$ for two agents, starting at potentially distinct depots, is lower-bounded by the cost C of covering the static G from these same depots. C can be computed in $O(n^3)$ via the algorithm described in Theorem 6. By following in $\mathcal{B}$ the journeys resulting in static cost C, each agent will wait at most $\Delta - 1$ steps for each successive edge to appear, thereby completing coverage in no more than ΔC steps. Since C is the fastest possible cost of covering G, this must be a Δ-approximation.

It is straightforward to extend to the case of k agents the result from [2] that DMVP for a single agent in $\mathcal{B}$ over trees is NP-hard to approximate within any factor less than Δ; simply link together by long paths k copies of the graph constructed for that proof. □

Over general graphs in $\mathcal{B}$, we can use spanning tree coverage to get the following approximation:

Theorem 9. *DMVP for two agents in $\mathcal{B}$ can be $\frac{12\Delta}{5}$-approximated in $O(n)$ time $\forall\ \Delta > 1$.*

Proof. Given a graph G, and a spanning tree H of G (constructed in $O(n)$ time), the Euler tour of H is a $2n-1$ node cycle C, the complete coverage of which implies complete coverage of G. From [1], for a cycle, we know each agent covers no more than $\lceil \frac{3}{5} \rceil |C| - 2$ edges in an optimal two agent solution, which can be found in $O(n)$ time. Following this solution in $\mathcal{B}$, the two agents take at most $\Delta(\lceil \frac{6n-3}{5} \rceil - 2)$ steps to complete coverage, which is no more than $\frac{12\Delta}{5}$ times worse than the minimum possible number of steps $\lceil \frac{n-1}{2} \rceil$ for covering G. □

We are able to extend Theorems 6 and 7 to any fixed number of agents k, applying ideas from the extension of 2-partition to k-partition for multisets of integers. With multiple depots, the union of the shortest paths between depots forms a k-leaf tree H. The possible costs of partitioning subtrees rooted at vertices in H but edge-disjoint from H and covering these subtrees along a path between two depots can be computed in a similar manner to the proof of Theorem 7. Then, the method of optimally ordering 01, 10, and 11 branches can be used on H itself. The methods for establishing approximation bounds for Theorem 8 will also still hold in the k agent case. For Theorem 9, bounds for cycle coverage in [1] enable $k\Delta$-approximations in $O(kn^3)$ for any $k > 2$. We will give the complete proofs in the full version of this paper.

4 Conclusion and Discussion

This paper has demonstrated the use of time-varying graphs for modeling multi-robot foremost coverage in dynamic environments, through consideration of the Dynamic Map Visitation Problem (DMVP). We have presented efficient algorithms for an arbitrary fixed number of agents for a range of topologies motivated by border coverage, and for two agents on a tree. Future work will extend Theorems 6, 7, 8 and 9 to a polynomial time solution for any fixed k, and we believe

it is also possible to make the extension to fixed p in Theorem 7. This begins by extending the idea that "when $p = 2$, an agent can enter any subtree at most once" to "for any $p > 1$, an agent at o can visit a node at depth $p - 1$ in G^o and return to o at most once".

In general, allowing for the number of agents to not be fixed increases the complexity of the problem, but when the number of agents becomes linear in the size of the graph—or, in the case of trees, linear in the number of leaves—special behavior can occur that further exposes the implications of applying constraints on edge dynamics and the number of depots to this type of problem. We make the following observation:

Remark 1. DMVP for $\frac{n}{c}$ agents on an star can be solved in polynomial time in $\mathcal{B}$ for any fixed Δ, but is hard in $\mathcal{R}$, for c constant.

Proof. Recall that a star is a tree in which at most one vertex has degree greater than 1. In $\mathcal{B}$, DMVP is upper-bounded by $2\Delta c$, when each agent is assigned c vertices to cover, and the foremost journeys taken between each are as long as possible. At each time step, each agent has $O(n)$ ways to continue its journey, so computing all possible journeys of time $\geq 2\Delta c$ for all agents takes $O((\frac{n^2}{c})^{2\Delta c}) = O(n^{4\Delta c})$. In $\mathcal{R}$, we cannot upper-bound the length of solutions, so all agents except one may be trapped together at a single vertex indefinitely, while the remaining agent is left alone to cover the rest of the star itself, which is NP-hard [2]. □

More generally, DMVP becomes tractable whenever it is possible to upper-bound optimal solutions by some constant, e.g., in $\mathcal{B}$ and $\mathcal{P}$ as k approaches n. This idea complements results of fixed parameter tractability for problems over TVGs of fixed treewidth [24], in which T is a fixed parameter.

Remark 2. With a single depot, DMVP for $k \geq m$ agents on an m-leaf tree is easy in $\mathcal{R}$; but with two depots, it is hard in $\mathcal{B}$, for all Δ.

Proof. If all agents start at a single depot s, sending one agent to each leaf l via the foremost journey from s to l will be optimal, even in $\mathcal{R}$. Now, consider the situation in $\mathcal{B}$, with $\Delta = 2$ over a spider with one sufficiently long leg. If one agent a starts at the center of the spider and the rest at the end of the long leg, in an optimal solution, a must cover all of the other legs before any other agent reaches the center, so the problem reduces to a single agent on a spider, which we know is hard. □

Decisive factors for DMVP tractability include environment topology, number of robots, and also the number of depots. The challenges of intractability that arise from these generalizations motivate research into online solutions to the problem. As a related example, [13] includes online approaches to static tree exploration with limited communication between agents. In future work, we plan to extend our results to markovian TVG models (e.g., [3,9]), which could support online solutions for general cases of map visitation problems in probabilistic dynamic environments.

References

1. Aaron, E., Kranakis, E., Krizanc, D.: On the complexity of the multi-robot, multi-depot map visitation problem. In: IEEE MASS, pp. 795–800 (2011)
2. Aaron, E., Krizanc, D., Meyerson, E.: DMVP: Foremost waypoint coverage of time-varying graphs. In: Kratsch, D., Todinca, I. (eds.) WG 2014. LNCS, vol. 8747, pp. 29–41. Springer, Heidelberg (2014). http://www.univ-orleans.fr/lifo/evenements/WG2014/
3. Baumann, H., Crescenzi, P., Fraigniaud, P.: Parsimonious flooding in dynamic graphs. Distr. Comp. **24**(1), 31–44 (2011)
4. Bektas, T.: The multiple traveling salesman problem: an overview of formulations and solution procedures. OMEGA **34**(3), 209–219 (2006)
5. Bhadra, S., Ferreira, A.: Complexity of connected components in evolving graphs and the computation of multicast trees in dynamic networks. In: Pierre, S., Barbeau, M., An, H.-C. (eds.) ADHOC-NOW 2003. LNCS, vol. 2865, pp. 259–270. Springer, Heidelberg (2003)
6. Bui-Xuan, B., Ferreira, A., Jarry, A.: Computing shortest, fastest, and foremost journeys in dynamic networks. IJ Found. Comp. Sci. **14**(02), 267–285 (2003)
7. Casteigts, A., Flocchini, P., Mans, B., Santoro, N.: Deterministic computations in time-varying graphs: broadcasting under unstructured mobility. In: Calude, C.S., Sassone, V. (eds.) TCS 2010. IFIP AICT, vol. 323, pp. 111–124. Springer, Heidelberg (2010)
8. Casteigts, A., Flocchini, P., Quattrociocchi, W., Santoro, N.: Time-varying graphs and dynamic networks. IJPED **27**(5), 387–408 (2012)
9. Avin, C., Koucký, M., Lotker, Z.: How to explore a fast-changing world (Cover Time of a Simple Random Walk on Evolving Graphs). In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part I. LNCS, vol. 5125, pp. 121–132. Springer, Heidelberg (2008)
10. Choset, H.: Coverage for robotics - a survey of recent results. Ann. Math. Artif. Intell. **31**, 113–126 (2001)
11. Correll, N., Rutishauser, S., Martinoli, A.: Comparing coordination schemes for miniature robotic swarms: a case study in boundary coverage of regular structures. In: Khatib, O., Kumar, V., Rus, D. (eds.) Experimental Robotics. STAR, vol. 39, pp. 471–480. Springer, Heidelberg (2008)
12. Dynia, M., Korzeniowski, M., Schindelhauer, C.: Power-aware collective tree exploration. In: Grass, W., Sick, B., Waldschmidt, K. (eds.) ARCS 2006. LNCS, vol. 3894, pp. 341–351. Springer, Heidelberg (2006)
13. Dynia, M., Kutyłowski, J., der Heide, F.M., Schindelhauer, C.: Smart robot teams exploring sparse trees. In: Královič, R., Urzyczyn, P. (eds.) MFCS 2006. LNCS, vol. 4162, pp. 327–338. Springer, Heidelberg (2006)
14. Easton, K., Burdick, J.: A coverage algorithm for multi-robot boundary inspection. In: Proceedings of ICRA, pp. 727–734 (2005)
15. Fakcharoenphol, J., Harrelson, C., Rao, S.: The k-traveling repairman problem. ACM Trans. Algorithms **3**(4) (2007). http://dl.acm.org/citation.cfm?doid=1290672.1290677
16. Flocchini, P., Mans, B., Santoro, N.: On the exploration of time-varying networks. Theor. Comput. Sci. **469**, 53–68 (2013)
17. Garey, M., Johnson, D.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman, New York (1979)

18. Godard, E., Mazauric D.: Computing the dynamic diameter of non-deterministic dynamic networks is hard. In: Gao, J., Efrat, A., Fekete, S.P., Zhang, Y. (eds.) ALGOSENSORS 2014. LNCS, vol. 8847, pp. 88–102. Springer, Heidelberg (2015)
19. Ilcinkas, D., Wade, A.M.: On the power of waiting when exploring public transportation systems. In: Fernàndez Anta, A., Lipari, G., Roy, M. (eds.) OPODIS 2011. LNCS, vol. 7109, pp. 451–464. Springer, Heidelberg (2011)
20. Ilcinkas, D., Wade, A.M.: Exploration of the T-Interval-Connected Dynamic Graphs: the case of the ring. In: Moscibroda, T., Rescigno, A.A. (eds.) SIROCCO 2013. LNCS, vol. 8179, pp. 13–23. Springer, Heidelberg (2013)
21. Kuhn, F., Lynch, N., Oshman, R.: Distributed computation in dynamic networks. In: STOC, pp. 513–522 (2010)
22. Kuhn, F., Oshman, R.: Dynamic networks: models and algorithms. ACM SIGACT News **42**(1), 82–96 (2011)
23. Kumar, S., Lai, T., Arora, A.: Barrier coverage with wireless sensors. In: ACM MobiCom, pp. 284–298 (2005)
24. Mans, B., Mathieson, L.: On the treewidth of dynamic graphs. In: Du, D.-Z., Zhang, G. (eds.) COCOON 2013. LNCS, vol. 7936, pp. 349–360. Springer, Heidelberg (2013)
25. Michail, O., Spirakis, P.G.: Traveling salesman problems in temporal graphs. In: Csuhaj-Varjú, E., Dietzfelbinger, M., Ésik, Z. (eds.) MFCS 2014, Part II. LNCS, vol. 8635, pp. 553–564. Springer, Heidelberg (2014)
26. Nagamochi, H., Okada, K.: A faster 2-approximation algorithm for the minmax p-traveling salesmen problem on a tree. Discrete Applied Math. **140**(1-3), 103–114 (2004)
27. Xu, L., Xu, Z., Xu, D.: Exact and approximation algorithms for the minmax k-traveling salesmen problem on a tree. EJOR **227**, 284–292 (2013)