

Parameterized Algorithms for Coordinated Motion Planning: Minimizing Energy

Argyrios Deligkas 

Department of Computer Science, Royal Holloway, University of London, Egham, UK

Eduard Eiben 

Department of Computer Science, Royal Holloway, University of London, Egham, UK

Robert Ganian 

Algorithms and Complexity Group, TU Wien, Vienna, Austria

Iyad Kanj 

School of Computing, DePaul University, Chicago, USA

M. S. Ramanujan 

Department of Computer Science, University of Warwick, UK

Abstract

We study the parameterized complexity of a generalization of the coordinated motion planning problem on graphs, where the goal is to route a specified subset of a given set of k robots to their destinations with the aim of minimizing the total energy (i.e., the total length traveled). We develop novel techniques to push beyond previously-established results that were restricted to solid grids.

We design a fixed-parameter additive approximation algorithm for this problem parameterized by k alone. This result, which is of independent interest, allows us to prove the following two results pertaining to well-studied coordinated motion planning problems: (1) A fixed-parameter algorithm, parameterized by k , for routing a single robot to its destination while avoiding the other robots, which is related to the famous Rush-Hour Puzzle; and (2) a fixed-parameter algorithm, parameterized by k plus the treewidth of the input graph, for the standard COORDINATED MOTION PLANNING (CMP) problem in which we need to route all the k robots to their destinations. The latter of these results implies, among others, the fixed-parameter tractability of CMP parameterized by k on graphs of bounded outerplanarity, which include bounded-height subgrids.

We complement the above results with a lower bound which rules out the fixed-parameter tractability for CMP when parameterized by the total energy. This contrasts the recently-obtained tractability of the problem on solid grids under the same parameterization. As our final result, we strengthen the aforementioned fixed-parameter tractability to hold not only on solid grids but all graphs of bounded local treewidth – a class including, among others, all graphs of bounded genus.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases coordinated motion planning, multi-agent path finding, parameterized complexity.

Digital Object Identifier 10.4230/LIPIcs...

Funding *Argyrios Deligkas*: Supported by Engineering and Physical Sciences Research Council (EPSRC) grant EP/X039862/1.

Robert Ganian: Project No. Y1329 of the Austrian Science Fund (FWF), Project No. ICT22-029 of the Vienna Science Foundation (WWTF).

Iyad Kanj: DePaul URC Grants 606601 and 350130

M. S. Ramanujan: Supported by Engineering and Physical Sciences Research Council (EPSRC) grants EP/V007793/1 and EP/V044621/1.



© A. Deligkas, E. Eiben, R. Ganian, I. Kanj, R. M. Sridharan;
licensed under Creative Commons License CC-BY 4.0

Leibniz International Proceedings in Informatics

LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

The task of routing a set of robots (or agents) from their initial positions to specified destinations while avoiding collisions is of great importance in a multitude of different settings. Indeed, there is an extensive body of works dedicated to algorithms solving this task, especially in the computational geometry [1, 24, 25, 27, 31, 33–35, 38], artificial intelligence [5, 36, 39] and robotics [3, 23, 37] research communities. Such algorithms typically aim at providing a schedule for the robots which is not only safe (in the sense of avoiding collisions), but which also optimizes a certain measure of the schedule – typically its makespan or energy (i.e., the total distance traveled by all robots). In this article, we focus solely on the task of optimizing the latter of these two measures.

A common formalization of our task of interest is given by the COORDINATED MOTION PLANNING problem (also known as MULTIAGENT PATHFINDING). In the context of energy minimization, the problem can be stated as follows: Given a graph G , a budget ℓ and a set $\mathcal{M}$ of robots, each equipped with an initial vertex and destination vertex in G , compute a schedule which delivers all the robots to their destinations while ensuring that the combined length traveled of all the robots is at most ℓ . Here, a schedule can be seen as a sequence of commands to the robots, where at every time step each of the robots can move to an adjacent vertex as long as no vertex or edge is used by more than a single robot at that time step. COORDINATED MOTION PLANNING, which generalizes the famous NP-hard $(n^2 - 1)$ -puzzle [10, 30], has been extensively studied—both in the discrete setting considered here as well as in various continuous settings [4, 5, 9, 13, 23, 29, 33–37, 39–41]—and has also been the target of specific computing challenges [16]. In other variants of the problem, there is no requirement to route *all* the robots to their destinations – sometimes the majority (or all but 1) of the robots are simply movable obstacles that do not have destinations of their own. This is captured, e.g., by the closely-related and well-studied RUSH HOUR puzzle/problem [6, 18] and by GRAPH MOTION PLANNING WITH 1 ROBOT (GCMP1) [28], which both feature a single robot with a designated destination.

Both COORDINATED MOTION PLANNING and GCMP1 are known to be NP-hard [10, 30, 40]. While several works have already studied COORDINATED MOTION PLANNING in the context of approximation and classical complexity theory, a more fine-grained investigation of the difficulty of finding optimal schedules through the lens of *parameterized complexity* [8, 12] was only carried out recently [14, 17]. The work in [14] established the fixed-parameter tractability¹ of COORDINATED MOTION PLANNING parameterized by either the number k of robots or the budget ℓ (as well as the makespan variant when parameterized by k), but only on solid grids; a *solid* grid is a standard rectangular $p \times q$ grid (i.e., with no holes), for some $p, q \in \mathbb{N}$. The more recent work in [17] showed the W[1]-hardness of the makespan variant of COORDINATED MOTION PLANNING w.r.t. the number of robots. The paper [17] also showed the NP-hardness of the makespan problem-variant on trees, and presented parameterized complexity results with respect to several combinations of parameters. In this article, we focus our attention on GCMP [28], which generalizes both COORDINATED MOTION PLANNING and GCMP1 by allowing an arbitrary partitioning of the robots into those with destinations and those which act merely as movable obstacles².

¹ A problem is *fixed-parameter tractable* w.r.t. a parameter k if it can be solved in time $f(k) \cdot n^{\mathcal{O}(1)}$, where n is the input size and f is a computable function.

² While previous hardness results for GCMP considers serial motion of robots (e.g., [28]), the hardness applies to parallel motion as well. Here we obtain algorithms for the coordinated motion variant with parallel movement, but note that the results can be directly translated to the serial version.

Contribution. The aim of this article is to push our understanding of the parameterized complexity of finding energy-optimal schedules beyond the class of solid grids. While this aim was already highlighted in the aforementioned paper on solid grids [14, Section 5], the techniques used there are highly specific to that setting and it is entirely unclear how one could generalize them even to the setting of subgrids; a *subgrid* is a subgraph of a solid grid and we define its *height* to be the minimum of its two dimensions.

As our two main contributions, we provide novel fixed-parameter algorithms (1) for GCMP1 parameterized by the number of robots alone (Theorem 1), and (2) for GCMP parameterized by the number of robots plus the treewidth of the graph (Theorem 2). Theorem 2 implies, as a corollary, the fixed-parameter tractability of GCMP parameterized by k plus the minimum dimension of the subgrid.

► **Theorem 1.** *GCMP1 is FPT parameterized by the number k of robots.*

► **Theorem 2.** *GCMP is FPT parameterized by the number of robots and the treewidth of the input graph.*

The main technical tool we use to obtain both of these results is a novel fixed-parameter approximation algorithm for GENERALIZED COORDINATED MOTION PLANNING parameterized by k alone, where the approximation error is only *additive* in k . We believe this result – summarized in Theorem 3 below – to be of independent interest.

► **Theorem 3.** *There is an FPT approximation algorithm for GCMP parameterized by the number k of robots which guarantees an additive error of $\mathcal{O}(k^5)$.*

The proof of Theorem 1 builds upon Theorem 3. For the proof of Theorem 2, we need to combine the approximation algorithm with novel insights concerning the “decomposability” of schedules along small separators in order to design a treewidth-based dynamic programming algorithm for the problem. A brief summary of the ideas used in this proof is provided at the beginning of Section 3.

We complement our positive results which use k as a parameter with an algorithmic lower bound showing that COORDINATED MOTION PLANNING is W[1]-hard (and hence not fixed-parameter tractable under well-established complexity-theoretic assumptions) when parameterized by the energy budget ℓ . This result (Theorem 4 below) contrasts the fixed-parameter tractability of the problem under the same parameterization when restricted to solid grids [14, Theorem 19].

► **Theorem 4.** *GCMP is W[1]-hard when parameterized by ℓ .*

While Theorem 4 establishes the intractability of the problem when parameterized by the energy on general graphs, one can in fact show that GCMP is fixed-parameter tractable under the same parameterization when the graphs are “well-structured” in the sense of having bounded *local treewidth* [15, 22]. This implies fixed-parameter tractability, e.g., on graphs of bounded genus and generalizes the aforementioned result on grids [14, Theorem 19].

► **Theorem 5.** *GCMP is FPT parameterized by ℓ on graph classes of bounded local treewidth.*

Further Related Work. As surveyed above, the computational complexity of GCMP has received significant attention, particularly by researchers in the fields of computational geometry, AI/Robotics, and theoretical computer science. The problem has been shown to remain NP-hard under a broad set of restrictions, including on graphs where only a single

vertex is not occupied [21], on grids [3], and bounded-height subgrids [20]. On the other hand, a feasibility check for the existence of a schedule can be carried out in polynomial time [42]. The recent AAMAS blue sky paper [32] also highlighted the need of understanding the hardness of the problem and asked for a deeper investigation of its parameterized complexity.

2 Terminology and Problem Definition

The graphs considered in this paper are undirected simple graphs. We assume familiarity with the standard graph-theoretic concepts and terminology [11]. For a subgraph H of a graph G and two vertices $u, v \in V(H)$, we denote by $\text{dist}_H(u, v)$ the length of a shortest path in H between u and v . We write $[n]$, where $n \in \mathbb{N}$, for $\{1, \dots, n\}$.

Parameterized Complexity. A *parameterized problem* Q is a subset of $\Omega^* \times \mathbb{N}$, where Ω is a fixed alphabet. Each instance of Q is a pair (I, κ) , where $\kappa \in \mathbb{N}$ is called the *parameter*. A parameterized problem Q is *fixed-parameter tractable* (FPT) [8, 12, 19], if there is an algorithm, called an *FPT-algorithm*, that decides whether an input (I, κ) is a member of Q in time $f(\kappa) \cdot |I|^{\mathcal{O}(1)}$, where f is a computable function and $|I|$ is the input instance size. The class FPT denotes the class of all fixed-parameter tractable parameterized problems.

Showing that a parameterized problem is hard for the complexity classes $W[1]$ or $W[2]$ rules out the existence of a fixed-parameter algorithm under well-established complexity-theoretic assumptions. Such hardness results are typically established via a *parameterized reduction*, which is an analogue of a classical polynomial-time reduction with two notable distinctions: a parameterized reduction can run in time $f(k) \cdot n^{\mathcal{O}(1)}$, but the parameter of the produced instance must be upper-bounded by a function of the parameter in the original instance. We refer to [8, 12] for more information on parameterized complexity.

Treewidth. Treewidth is a structural parameter that provides a way of expressing the resemblance of a graph to a forest. Formally, the treewidth of a graph is defined via the notion of tree decompositions as follows.

► **Definition 6 (Tree decomposition).** A *tree decomposition* of a graph G is a pair (T, β) of a tree T and $\beta : V(T) \rightarrow 2^{V(G)}$, such that:

- $\bigcup_{t \in V(T)} \beta(t) = V(G)$,
- for any edge $e \in E(G)$, there exists a node $t \in V(T)$ such that both endpoints of e belong to $\beta(t)$, and
- for any vertex $v \in V(G)$, the subgraph of T induced by the set $T_v = \{t \in V(T) : v \in \beta(t)\}$ is a tree.

The *width* of (T, β) is $\max_{v \in V(T)} \{|\beta(v)|\} - 1$. The *treewidth* of G is the minimum width of a tree decomposition of G .

Let (T, β) be a tree decomposition of a graph G . We refer to the vertices of the tree T as *nodes*. We always assume that T is a rooted tree and hence, we have a natural parent-child and ancestor-descendant relationship among nodes in T . A *leaf* node nor a *leaf* of T is a node with degree exactly one in T which is different from the root node. All the nodes of T which are neither the root node or a leaf are called *non-leaf* nodes. The set $\beta(t)$ is called the *bag* at t . For two nodes $u, t \in T$, we say that u is a *descendant* of t , denoted $u \preceq t$, if t lies on the unique path connecting u to the root. Note that every node is its own descendant. If $u \preceq t$ and $u \neq t$, then we write $u \prec t$. For a tree decomposition (T, β) we also have a mapping $\gamma : V(T) \rightarrow 2^{V(G)}$ defined as $\gamma(t) = \bigcup_{u \prec t} \beta(u)$.

We use the following structured tree decomposition in our algorithm.

► **Definition 7 (Nice tree decomposition).** Let (T, β) be a tree decomposition of a graph G , where r is the root of T . The tree decomposition (T, β) is called a *nice tree decomposition* if the following conditions are satisfied.

1. $\beta(r) = \emptyset$ and $\beta(\ell) = \emptyset$ for every leaf node ℓ of T ; and
2. every non-leaf node (including the root node) t of T is of one of the following types:
 - **Introduce node:** The node t has exactly one child t' in T and $\beta(t) = \beta(t') \cup \{v\}$, where $v \notin \beta(t')$.
 - **Forget node:** The node t has exactly one child t' in T and $\beta(t) = \beta(t') \setminus \{v\}$, where $v \in \beta(t')$.
 - **Join node:** The node t has exactly two children t_1 and t_2 in T and $\beta(t) = \beta(t_1) = \beta(t_2)$.

Efficient fixed-parameter algorithms are known for computing a nice tree-decomposition of near-optimal width:

► **Proposition A ([26]).** *There exists an algorithm which, given an n -vertex graph G and an integer k , in time $2^{\mathcal{O}(k)} \cdot n$ either outputs a nice tree-decomposition of G of width at most $2k + 1$ and $\mathcal{O}(n)$ nodes, or determines that $\text{tw}(G) > k$.*

Boundaried graphs. Roughly speaking, a boundaried graph is a graph where some vertices are annotated. A formal definition is as follows.

► **Definition B (Boundaried Graph).** A *boundaried graph* is a graph G with a set $X \subseteq V(G)$ of distinguished vertices called *boundary vertices*. The set X is the *boundary* of G . A boundaried graph (G, X) is called a *p-boundaried graph* if $|X| \leq p$.

► **Definition C.** A *p-boundaried subgraph* of a graph G is a *p-boundaried graph* (H, Z) such that (i) H is a vertex-induced subgraph of G and Z separates $V(H) \setminus Z$ from $V(G) \setminus V(H)$.

- **Definition D.** Let G be a graph with a tree decomposition (T, β) and let $x \in V(T)$.
- $G_{x,T}^\uparrow$ denotes the boundaried graph obtained by taking the subgraph of G induced by the set $V(G) \setminus (\gamma(x) \setminus \beta(x))$, with $\beta(x)$ as the boundary.
 - Similarly, $G_{x,T}^\downarrow$ denotes the boundaried graph obtained by taking the subgraph of G induced by $\gamma(x)$ with $\beta(x)$ as the boundary.

Notice that if G is a graph with a tree decomposition (T, β) of width at most k and $x \in T$, then $G_{x,T}^\uparrow$ and $G_{x,T}^\downarrow$ are both $(k + 1)$ -boundaried graphs.

Problem Definition. In our problems of interest, we are given an undirected graph G and a set $\mathcal{R} = \{R_1, R_2, \dots, R_k\}$ of k robots where $\mathcal{R}$ is partitioned into two sets $\mathcal{M}$ and $\mathcal{F}$. Each $R_i \in \mathcal{M}$ has a starting vertex s_i and a destination vertex t_i in $V(G)$ and each $R_i \in \mathcal{F}$ is associated only with a starting vertex $s_i \in V(G)$. We refer to the elements in the set $\{s_i \mid i \in [k]\} \cup \{t_i \mid R_i \in \mathcal{M}\}$ as *terminals*. The set $\mathcal{M}$ contains robots that have specific destinations they must reach, whereas $\mathcal{F}$ is the set of remaining “free” robots. We assume that all the s_i are pairwise distinct and that all the t_i are pairwise distinct. At each time step, a robot may either move to an adjacent vertex, or stay at its current vertex, and robots may move simultaneously. We use a discrete time frame $[0, t]$, $t \in \mathbb{N}$, to reference the sequence of moves of the robots and in each time step $x \in [0, t]$ every robot remains stationary or moves.

A *route* for robot R_i is a tuple $W_i = (u_0, \dots, u_t)$ of vertices in G such that (i) $u_0 = s_i$ and $u_t = t_i$ if $R_i \in \mathcal{M}$ and (ii) $\forall j \in [t]$, either $u_{j-1} = u_j$ or $u_{j-1}u_j \in E(G)$. Put simply, W_i corresponds to a “walk” in G , with the exception that consecutive vertices in W_i may be identical (representing waiting time steps), in which R_i begins at its starting vertex at

time step 0, and if $R_i \in \mathcal{M}$ then R_i reaches its destination vertex at time step t . Two routes $W_i = (u_0, \dots, u_t)$ and $W_j = (v_0, \dots, v_t)$, where $i \neq j \in [k]$, are *non-conflicting* if (i) $\forall r \in \{0, \dots, t\}$, $u_r \neq v_r$, and (ii) $\nexists r \in \{0, \dots, t-1\}$ such that $v_{r+1} = u_r$ and $u_{r+1} = v_r$. Otherwise, we say that W_i and W_j *conflict*. Intuitively, two routes conflict if the corresponding robots are at the same vertex at the end of a time step, or go through the same edge (in opposite directions) during the same time step.

A *schedule* S for $\mathcal{R}$ is a set of pairwise non-conflicting routes $W_i, i \in [k]$, during a time interval $[0, t]$. The (*traveled*) *length* of a route (or its associated robot) within S is the number of time steps j such that $u_j \neq u_{j+1}$. The *total traveled length* of a schedule is the sum of the lengths of its routes; this value is often called the *energy* in the literature (e.g., see [16]).

Using the introduced terminology, we formalize the GENERALIZED COORDINATED MOTION PLANNING WITH ENERGY MINIMIZATION (GCMP) problem below.

GCMP

Input: A tuple $(G, \mathcal{R} = (\mathcal{M}, \mathcal{F}), k, \ell)$, where G is a graph, $\mathcal{R} = \{R_i \mid i \in [k]\}$ is a set of robots partitioned into sets $\mathcal{M}$ and $\mathcal{F}$, where each robot in $\mathcal{M}$ is given as a pair of vertices (s_i, t_i) and each robot in $\mathcal{F}$ as a single vertex s_i , and $k, \ell \in \mathbb{N}$.

Problem: Is there a schedule for $\mathcal{R}$ of total traveled length at most ℓ ?

By observing that the feasibility check of Yu and Rus [42] transfers seamlessly to the case where some robots do not have destinations, we obtain the following.

► **Proposition 8** ([42]). *The existence of a schedule for an instance of GCMP can be decided in linear time. Moreover, if such a schedule exists, then a schedule with total length traveled of $\mathcal{O}(|V(G)|^3)$ can be computed in $\mathcal{O}(|V(G)|^3)$ time.*

Proposition 8 implies inclusion in NP, and allows us to assume henceforth that every instance of GCMP is feasible (otherwise, in linear time we can reject the instance). We denote by GCMP1 the restriction of GCMP to instances where $|\mathcal{M}| = 1$. We remark that even though GCMP is stated as a decision problem, all the algorithms provided in this paper are constructive and can output a corresponding schedule (when it exists) as a witness.

3 An Additive FPT Approximation for GCMP

In this section, we give an FPT approximation algorithm for GCMP with an additive error that is a function of the number k of robots.

We start by providing a high-level, low-rigor intuition for the main result of this section. Let $\mathcal{I} = (G, \mathcal{R} = (\mathcal{M}, \mathcal{F}), k, \ell)$ be an instance of GCMP. Ideally, we would like to route the robots in $\mathcal{M}$ along shortest paths to their destinations, while having the other robots move away only a “little” ($\epsilon(k)$ -many steps) to unblock the shortest paths for the robots in $\mathcal{M}$. Unfortunately, this might not be possible as it is easy to observe that a free robot might have to travel a long distance in order to unblock the shortest paths of other robots. (For instance, think about the situation where a free robot is positioned in the middle of a long path of degree-2 vertices that the shortest paths traverse.) However, we will show that such a situation could only happen if the blocking robots are positioned in simple graph structures containing a long path of degree-2 vertices.

We then exploit these simple structures to “guess” in FPT-time, for each robot, a location which it visits in an optimal solution and which is in the vicinity of a safe location, called a “haven”; this haven is centered around a “nice” vertex of degree at least 3, and allows the

robot to avoid any passing robot within $\epsilon(k)$ moves. We show how to navigate the robots to these havens optimally in the case of GCMP1, and with an $\epsilon(k)$ overhead in the case of GCMP. Moreover, this navigation is well-structured and can be leveraged to show that no robot in an optimal solution will visit the same vertex many times. A similar navigation takes place at the end, during the routing of the robots from their havens to their destinations.

Once we obtain such a reduced instance in which all starting positions and destinations of the robots are in havens, we can use our intended strategy to navigate each robot in $\mathcal{M}$ along a shortest path, with only an $\epsilon(k)$ overhead, which immediately gives us the approximation result and the property that no robot visits any vertex more than $\epsilon(k)$ times in an optimal solution. This latter property about the optimal solution is crucial, as we exploit it later in Section 5 to design an intricate dynamic programming algorithm over a tree decomposition of the input graph.

In addition, each free robot moves at most $\epsilon(k)$ times in the considered solution for the reduced instance. This, together with the equivalence between $\mathcal{I}$ and the reduced instance in the case of GCMP1, allow us to restrict the movement of the free robots in an optimal solution to only $\epsilon(k)$ -many locations, which we use in Section 4 to obtain the FPT algorithm for GCMP1.

We start by defining the notion of a nice vertex and its haven.

► **Definition 9** (Nice Vertex). A vertex $w \in V(G)$ is *nice* if there exist three connected subgraphs C_1, C_2, C_3 of G such that: (i) the pairwise intersection of the vertex sets of any pair of these subgraphs is exactly w , and (ii) $|V(C_1)| \geq k + 1$, $|V(C_2)| \geq k + 1$, and $|V(C_3)| \geq 2$. If w is nice, let $x \in V(C_3)$ be a neighbor of w , and define the *haven* H_w of w to be the subgraph of G induced by the vertices in $\{x\} \cup V(C_1) \cup V(C_2)$ whose distance from w in H_w (i.e., $\text{dist}_{H_w}(w, u)$, for $u \in V(C_1) \cup V(C_2)$) is at most k .

For a set $S \subseteq \mathcal{R}$ of robots and a subgraph H , a *configuration* of S w.r.t. H is an injection $\iota : S \rightarrow V(H)$. The following lemma shows that we can take the robots in a haven from any configuration to any other configuration in the haven, while incurring a total of $\mathcal{O}(k^3)$ travel (in the haven) length.

► **Lemma 10.** *Let w be a nice vertex, let C_1, C_2, C_3 be three subgraphs satisfying conditions (i) and (ii) of Definition 9, and let H_w be a haven for w . Then for any set of robots $S \subseteq \mathcal{R}$ in H_w with current configuration $\iota(S)$, any configuration $\iota'(S)$ with respect to H_w can be obtained from $\iota(S)$ via a sequence of $\mathcal{O}(k^3)$ moves that take place in H_w .*

Proof. Let T_1 (resp. T_2) be a Breadth-First Search (BFS) tree in H_w rooted at w whose vertex set is $V(H_w) \cap V(C_1)$ (resp. $V(H_w) \cap V(C_2)$). Note that T_1 and T_2 are well defined since C_1 and C_2 are connected. Moreover, we have $|V(T_1)|, |V(T_2)| \geq k + 1$, and w is the only intersection between the two sets $V(T_1)$ and $V(T_2)$. Let $x \in H_w$ be the neighbor of w in C_3 . To arrive at configuration $\iota'(S)$ from $\iota(S)$, we perform the following process.

In the first phase of this process, we move all the robots in S from their current vertices to arbitrary distinct vertices in $V(T_1) - \{w\}$. Note that this is possible since $|V(T_1) - \{w\}| \geq k$. Moreover, this is achievable in $\mathcal{O}(k^2)$ robot moves since every vertex in S is at distance at most k from w . For instance, one way of achieving the above is to move first the robots in S that are in T_1 , one by one to the deepest unoccupied vertex in T_1 , then repeat this for the robots in S that are in T_2 , and finally move the robot on x (if there is such a robot) to T_1 .

Let $S_2 = \{R \in S \mid \iota'(R) \in V(T_2) \setminus \{w\}\}$ (i.e., the set of robots in S whose final destinations w.r.t. $\iota'(S)$ are in T_2 excluding vertex w). In the second phase, we route the robots in S_2 in reverse order of the depths of their destinations in T_2 ; that is, for two robots R and R' in S_2 ,

if R 's destination is deeper than that of R' , then R is routed first; ties are broken arbitrarily. To route a robot $R \in S_2$, we first move R in T_1 towards w as follows. Let P be the unique path in T_1 from w to the vertex at which R currently is. (Note that P may contain other robots.) We “pull” the robots along P towards w , and into T_2 until R reaches w , and then move R to vertex x . When a robot $R' \neq R$ on P reaches w , R' is then routed towards an unoccupied vertex in T_2 by finding a root-leaf path P' in T_2 that contains an unoccupied vertex (which must exist since $|V(T_2)| \geq k + 1$), and “pushing” the robots on P' down that path, possibly pushing (other) robots on P' along the way, until R' and every robot on P' occupy distinct vertices in T_2 . Once R is moved to x , all the robots on P that were moved into T_2 during this step are moved back into T_1 (i.e., we reverse the process). Afterwards, R is routed to its destination in T_2 , along the unique path from w to that destination; note that this path must be devoid of robots due to the ordering imposed on the robots in the routing scheme. The above process is then repeated for every robot in S_2 , routing it to its location stipulated by $\iota'(S)$.

In the last phase, we move all the robots in $S_1 = S - S_2$, one by one, to T_2 by applying the same “sliding” operation as in the second phase. As a robot is moved to T_2 , it may push the robots in T_2 deeper into T_2 . Note that, during this process, no robot in T_2 occupies a vertex that is an ancestor of a robot S_1 , and hence no robot in T_2 “blocks” a robot in S_1 from coming back to T_1 . We then route the robots in S_1 from T_2 back into T_1 in the reverse order of the depths of their destinations w.r.t. $\iota'(S)$ in T_1 . However, if a robot has either w or x as a destination, then we route the robot whose destination is w last, and the one whose destination is x second from last. To route a robot $R \in S_1$ from T_2 to its destination in T_1 , we pull R up in T_2 towards w , possibly pulling along the way other robots (that are only in S_1) into T_1 , until R escapes to x . We then slide the robots that we pulled into T_1 back to T_2 and route R to its final destination in $T_1 \cup \{w, x\}$. We repeat this step until all the robots in S_1 have been routed to their destination in $T_1 \cup \{w, x\}$ as stipulated by $\iota'(S)$.

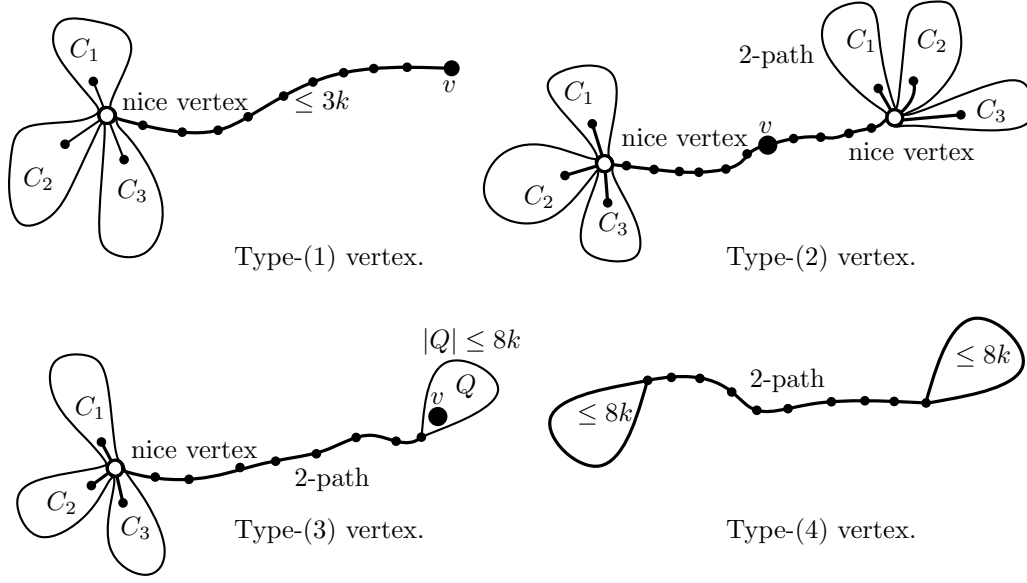
Since $|V(H)| = \mathcal{O}(k)$, it is not difficult to verify that the total traveled length by all the robots over the whole process is $\mathcal{O}(k^3)$. ◀

► **Lemma E.** *Let w be a vertex of degree at least 3 that is contained in a (simple) cycle C of length at least $2k + 2$. Then w is nice.*

Proof. Let $C = (w_0 = w, w_1, \dots, w_r)$, where $r \geq 2k + 1$. Let $x \notin \{w_1, w_r\}$ be a neighbor of w . If $x \notin V(C)$, then the subgraphs consisting of the single edge wx , and the two subpaths $(w, w_1, \dots, w_k)$ and $(w, w_r, \dots, w_{r-k+1})$ of C satisfy conditions (i) and (ii) in Definition 9, and w is nice.

Suppose now that $x \in V(C)$, and let x_1, x_2 be the neighbors of x that appear adjacently (on either side) to x on C . Let P_{wx_1} be the subpath of C between w and x_1 and P_{wx_2} that between w and x_2 . At least one of P_{wx_1}, P_{wx_2} contains at least $k + 1$ vertices; otherwise, C would have length at most $2k$. Assume, without loss of generality, that P_{wx_1} contains at least $k + 1$ vertices, and let P_w^- be the subpath of P_{wx_1} that starts at w and that contains exactly $k + 1$ vertices. Assume, without loss of generality, that w_1 is on P_w^- , and hence w_k is the endpoint of P_w^- other than w . Then it is easy to see that the three subgraphs consisting of the edge ww_r , the path P_w^- , and the connected subgraph consisting of $wx + P_{w_{r-1}w_k}$, where $P_{w_{r-1}w_k}$ is the subpath of C between w_{r-1} and w_k , satisfy conditions (i) and (ii) in Definition 9, implying that w is nice. ◀

Call a path P in G a *2-path* if P is an induced path of degree-2 vertices in G . The next lemma is used to characterize the possible graph structures around a vertex that is not nice; an illustration for the four cases handled by the lemma is provided in Figure 1.



■ **Figure 1** Illustration of the four types for a vertex v that is not nice.

► **Lemma 11.** *For every vertex v that is not nice, one of the following holds:*

- (1) *There is a nice vertex at distance at most $3k$ from v ; or*
- (2) *vertex v is on a 2-path whose both endpoints are nice vertices; or*
- (3) *there is a 2-path P such that one of its endpoints is nice and the other is contained in a connected subgraph Q of size at most $8k$ that P cuts from the nice vertex and such that v is in $P \cup Q$; or*
- (4) *Either $|V(G)| \leq 8k$, or G consists of a 2-path P , each of whose endpoints is contained in a subgraph of size at most $8k$, such that P cuts the two subgraphs from one another.*

Proof. Suppose that v is not a nice vertex, and assume that (1) does not hold (i.e., that there is no nice vertex at distance at most $3k$ from v). We will show that one of the Statements (2)-(4) must hold.

Let T be a BFS-tree of G rooted at v . For a vertex x in T , denote by $\deg_T(x)$ the degree of x in T , by T_x the subtree of T rooted at x , and let $T_x^- = T - V(T_x)$. For two vertices $x, y \in T$, denote by P_{xy} the unique path between x and y in T , and by $d_T(x, y)$ the distance between x and y in T , which is $|P_{xy}|$.

If there is no vertex x in T with $\deg_T(x) \geq 3$, then T is a path of vertices, each of degree at most 2 in T . Note that any edge in G is between two vertices that are at most one level apart in the BFS-tree T . If there is an edge $xy \in E(G) - E(T)$ that is between two vertices that are not at the same level of T , then assume w.l.o.g. that x has a higher level than y . We can “re-graft” y so that x becomes its parent in T (i.e., cut y from its current parent and attach it to x) to obtain another BFS-tree rooted at v and containing a vertex x with $\deg_T(x) \geq 3$. Suppose now that all the edges in $E(G) - E(T)$ join vertices at the same level of T . Let x and y be the two vertices in T of degree-3 in G that are of minimum distance to v , and note that $xy \in E(G)$. Moreover, $P_{xy} - x - y$ is a 2-path in G . If both x and y are nice, then v is contained in an induced path of degree-2 vertices in G that has two nice vertices at both ends (since no edges can exist between any two vertices along that path), and hence, v satisfies Statement (2) of the lemma. If one of x, y , say x is nice, then y must be

nice as well by Lemma E, since $d_T(v, x) = d_T(v, y) > 3k$ (as (1) does not hold), and hence y is a degree-3 vertex that is contained in the cycle $P_{xy} + xy$ of length at least $3k + 1$. Suppose now that both x and y are not nice. Observe in this case that $d_T(v, x) = d_T(v, y) \leq k + 1$; otherwise, x and y would be nice by Lemma E. If each of T_x, T_y contains more than $k + 1$ vertices, then again x and y would be nice. It follows that at least one of T_x, T_y , say T_x , contains at most k vertices, and hence G consists of a subgraph containing T_x, P_{xy} and at most k neighbors in T_y of the vertices in T_x that must be within distance k from y , followed by a 2-path (the rest of T_y) in G . Therefore, G consists of a subgraph of at most $4k + 2$ vertices plus a 2-path in G , and Statement (4) in the lemma is satisfied.

Therefore, we may assume henceforth that there is a vertex x in T satisfying $\deg_T(x) \geq 3$.

We first claim that any subtree of T that contains a vertex of degree at least 3 in T contains a vertex x satisfying $\deg_T(x) \geq 3$ and $|V(T_x^-)| \geq 3k$, otherwise the statement of the lemma is satisfied.

Let $T_{v'}$ be a subtree containing a vertex whose degree in T is at least 3, and suppose that there is no vertex $x \in T_{v'}$ satisfying $\deg_T(x) \geq 3$ and $|V(T_x^-)| \geq 3k$. Since there exists a vertex in $T_{v'}$ whose degree in T is at least 3, for any vertex x in $T_{v'}$ satisfying $\deg_T(x) \geq 3$, we have $|V(T_x^-)| < 3k$. Let x be a vertex in $T_{v'}$ satisfying $\deg_T(x) \geq 3$ and with the maximum $d_T(v, x)$. Since $|V(T_x^-)| < 3k$ and T_x^- is connected and contains v , it follows that $d_T(v, x) < 3k$, and hence $d_G(v, x) < 3k$. Since – by our assumption – (1) is not satisfied, there is no nice vertex within distance at most $3k$ from v , it follows that x is not nice. Moreover, by the choice of x , for every child y of x in T , T_y is a path of degree-2 (in T) vertices. Since x is not nice, at most one of the paths rooted at a child of x can contain more than k vertices; call such a path *long*. If no such long path exists, then T_x contains at most $4k$ vertices; otherwise, we can split T_x into two subgraphs C_2, C_3 , intersecting only at x and each containing at least $k + 1$ vertices. It follows in this case that the number of vertices in T , and hence in G , is $|V(T_x)| + |V(T_x^-)| \leq 7k$, and G satisfies Statement (4) of the lemma (with the 2-path being empty). Suppose now that a long path P_x of degree-2 vertices rooted at x exists. Then since x is not nice, $T_x - V(P_x)$ contains at most k vertices, and hence T can be decomposed into two parts P_x and $T - P_x$ that intersect only at x , where $|V(T - P_x)| \leq 4k$. Note that, by the property of a BFS-tree, no vertex on P_x at distance more than $3k$ from x can be a neighbor of a vertex in $V(T - P_x)$, and hence, any vertex on P_x of distance at least $3k$ from x is a cut-vertex that cuts G into a connected subgraph of size at most $7k$ plus a 2-path in G , thus satisfying the Statement (4) of the lemma.

Suppose now that any subtree of T that contains a vertex of degree at least 3 in T contains a vertex x satisfying $\deg_T(x) \geq 3$ and $|V(T_x^-)| \geq 3k$. Since there exists a vertex x satisfying $\deg_T(x) \geq 3$, let w be such a vertex satisfying $|V(T_w^-)| \geq 3k$ and such that no ancestor y of w in T satisfies $\deg_T(y) \geq 3$ and $|V(T_y^-)| \geq 3k$.

If w is nice, consider the path P_{wv} of T . If there exists an internal degree-3 vertex (in T) on P_{wv} , then let x be the farthest such vertex from v (i.e., closest such ancestor of w) on this path. By the choice of w , $|V(T_x^-)| < 3k$, and hence $v \in V(T_x^-)$ is contained in a subgraph of size at most $3k$ that is connected by a (possibly empty) path of vertices of degree 2 in T to w . Since w is nice and Statement (1) is not satisfied (by our assumption), we have $d_T(v, w) > 3k$. Since $|V(T_x^-)| < 3k$, by the properties of BFS-trees, no vertex in $V(T_x^-)$ could be adjacent in G to any vertex in T_w or to any vertex y on P_{xw} , the path of T between x and w , satisfying $d_T(v, y) > 3k$. Therefore, by picking any vertex u on P_{xw} whose distance from w is at most 1 (Which must exist), we have a cut-vertex in G that cuts G into a component of size at most $6k$ containing v , connected by a 2-path in G to a nice vertex w , as stated in Statement (3) of the lemma.

Suppose now that no vertex x satisfying $\deg_T(x) \geq 3$ exists along P_{vw} . Note that $|P_{vw}| > 3k$, since the distance between v and the nice vertex w is more than $3k$.

If $\deg_T(v) \geq 3$, then since v is not nice, it follows that the union of all the subtrees rooted at the children of v , except the subtree containing w , is at most $2k$, and hence, by a similar analysis to the above, it is easy to see that v is part of a connected subgraph of size at most $4k$ that is connected by a 2-path P (which is a subpath of P_{vw}) in G to a nice vertex w such that P cuts this subgraph from w , thus satisfying Statement (3) of the lemma.

Suppose now that $\deg_T(v)$ is 2, and hence $\deg_G(v) = 2$. Let w' be the child of v such that $T_{w'}$ contains w and v' be the other child of v (i.e. such that $T_{v'}$ does not contain w).

Let z be the first descendant of v in $T_{v'}$ such that $\deg_G(z) \geq 3$; if no such z exists, then v must be on a 2-path in G that ends a nice vertex w , thus satisfying Statement (3) of the lemma.

Suppose now that z exists. If z is nice, then $d_T(v, z) > 3k$. Observe that any neighbor of z in G must be within 1 level from that of z in T . Observe also that Since P_{vz} is an induced path of degree-2 vertices in G . If z has a neighbor x in G on P_{vw} , let x be the neighbor with the minimum $d_T(v, x)$. Then x must be nice since x is a degree-3 vertex on a cycle of length at least $2k + 2$ (see Lemma E). Moreover, by the choice of z and x , v is contained in an induced path of degree-2 vertices in G whose both endpoints are nice vertices (x and z), thus satisfying Statement (2) of the lemma. Similarly, if any vertex x on P_{vw} has a neighbor in G , which must be a vertex below z in $T_{v'}$, and hence x must be on a cycle of length at least $2k + 2$ and must be nice, and v satisfies Statement (2) of the lemma. Otherwise, v is contained in a 2-path in G that ends at nice vertices, namely z and w , again satisfying Statement (2) of the lemma.

Suppose now that z is not nice. Since the length of $P_{zv} + P_{vw}$ is more than $3k$ and z is not nice, the number of vertices in T_z is at most $2k$. Note that, since z is not nice, z cannot be contained in a cycle of length more than $2k + 1$, and hence, no edge in G exists between a vertex in T_w and a vertex in T_z . If no vertex in P_{vw} has a neighbor in $T_{v'}$ then v is contained in 2-path P that on one side ends with a nice vertex w , and on the other side with a connected subgraph T_z of size at most $2k$ such that P cuts T_z from w , thus satisfying Statement (3) of the lemma. Otherwise, any vertex x on P_{vw} with a neighbor in T_z must be within distance at most $k + 1$ from v (otherwise, z would be contained in a cycle of length at least $2k + 2$ and would be nice). It follows that v is contained in a connected subgraph of size at most $4k + 2$ that is connected by a 2-path P in G to a nice vertex w such that P cuts w from this subgraph containing v , and thus satisfying Statement (3) of the lemma.

Suppose now that w is not nice. Then since $|V(T_w^-)| \geq 3k$ by the choice of w , $|V(T_w)| \leq 2k$.

Consider the path P_{wv} of T . If there exists a degree-3 (in T) vertex on P_{wv} , then let x be the deepest such vertex (ancestor of w) on this path. By the choice of w , $|V(T_x^-)| < 3k$, and hence $v \in V(T_x^-)$ is contained in a subgraph of size at most $3k$. Observe that any vertex in $V(T_x^-)$ cannot have a neighbor on P_{xw} that is at distance more than $3k$ from x . Hence, either the total size of G is at most $8k$ (if a vertex in $V(T_x^-)$ has a neighbor in T_w), or G can be decomposed into a 2-path P , each of its endpoints is a contained in a connected subgraph of size at most $8k$ (one of the two subgraphs must contain v) such that P separates the two. In either of the two cases, v satisfies Statement (4) of the lemma.

Suppose now that no such x exists, then w is connected to v by a path of degree-2 vertices in T . By symmetry, we can assume that, for every child v' of v , either $T_{v'}$ does not contain a vertex x satisfying $\deg_T(x) \geq 3$, or contains a vertex w' that is not nice and satisfying $|V(T_{w'}^-)| \leq 2k$, $|V(T_{w'})| \geq 3k$, and is connected to v via a path of degree-2 vertices in T .

If $\deg_T(v) \leq 2$, then we can argue – similarly to the above – that v must satisfy the lemma. We summarize these arguments below for the sake of completeness. Let w' be the child of v such that $T_{w'}$ contains w and v' be the other child of v . Either $T_{v'}$ is a path of vertices, each of degree at most 2 in T , or there exists a vertex x in $T_{v'}$ such that $\deg_T(x) \geq 3$, x is not nice, x satisfies $|V(T_x)| \leq 2k$ and $|V(T_x^-)| \geq 3k$, and P_{xv} is a path of degree-2 vertices in T . If $|P_{vw}| \leq 2k$, then $|V(T_{w'})| \leq 4k$ and no vertex in $T_{w'}$ can be adjacent in G to any vertex in $T_{v'}$ whose depth in T is more than $4k$; it is easy to see in this case that Statement (4) of the lemma holds. Suppose now that $|P_{vw}| > 2k$ and observe that since w is not nice, no vertex in T_w can be adjacent to a vertex in $T_{v'}$; otherwise, w would be contained in a cycle of length at least $2k + 2$. Therefore, the only edges in G that could exist between $T_{w'}$ and $T_{v'}$ are edges between P_{vw} and $T_{v'}$. If no edge exists between $T_{w'}$ and $T_{v'}$, then Statement (4) of the lemma holds. Otherwise, let xy be the edge between a vertex x in P_{vw} and a vertex y in $T_{v'}$ such that x is of minimum depth in $T_{w'}$. If x is nice then y must be nice (since both would be contained in a cycle of length at least $2k + 2$) and Statement (2) of the lemma holds. Suppose now that both x and y are not nice, which implies that each has depth at most $k + 1$ in T . Moreover, either T_x or T_y must contain at most k vertices. In each of these two cases, it is easy to see that Statement (4) of the lemma holds.

Suppose now that v has at least three children in T . At most one of these children can satisfy that the subtree of T rooted at it has at least k vertices; otherwise, v would be nice. If no child v_i of v satisfies that $|V(T_{v_i})| \geq k$, then it is easy to see that $|V(G)| \leq 6k$, and hence Statement (4) of the lemma is satisfied, as otherwise v would be nice. Suppose now that exactly one child of v , say v_1 , is such that $|V(T_{v_1})| \geq k$, and note in this case that $|V(T) - V(T_{v_1})| \leq 2k$. If T_{v_1} contains a vertex x satisfying $\deg_T(x) \geq 3$, then T_{v_1} contains a vertex w that is not nice and satisfying $|V(T_w)| \leq 2k$, $|V(T_w^-)| \geq 3k$, and such that P_{vw} is a path of degree-2 vertices in T . Since $|V(T) - V(T_{v_1})| \leq 2k$, and hence, any vertex in $V(T) - V(T_{v_1})$ can be adjacent only to vertices on P that are within distance at most $2k$ from v (since T is a BFS-tree), either G has at most $6k$ vertices, or it consists of two connected subgraphs, each of size at most $4k$ that are connected by a 2-path of G (which is a subpath of P) that separates the two subgraphs, and Statement (4) of the lemma holds. Suppose now that T_{v_1} consists of a path P of vertices each of degree at most 2 in T . Again, by the properties of the BFS-tree T , any vertex in $V(T) - V(T_{v_1})$ can be adjacent to vertices on P that are within distance at most $2k$ from v . It follows in this case that G can be decomposed into a connected subgraph of size at most $4k$ that contains v and that is connected by a 2-path, and Statement (4) of the lemma holds. ◀

► **Definition 12 (Vertex Types).** Let v be a vertex in G that is not nice. We say that v is a *type-(i) vertex*, where $i \in \{1, \dots, 4\}$, if v satisfies Statement (i) in Lemma 11 but does not satisfy any Statement (j), where $j < i$ (if Statement (j) exists). Note that, by Lemma 11, every vertex v that is not nice must be a type-(i) vertex for some $i \in \{1, \dots, 4\}$.

The following lemma shows that we can restrict our attention to the case where there is no type-(4) vertex in G .

► **Lemma 13.** Let $\mathcal{I} = (G, \mathcal{R} = (\mathcal{M}, \mathcal{F}), k, \ell)$ be an instance of GCMP. If there exists a type-(4) vertex $v \in V(G)$ then $\mathcal{I}$ can be solved in time $\mathcal{O}^*((4k^2 + 16k)^{2k})$ and hence is FPT.

Proof. Suppose that there exists a type-(4) vertex $v \in V(G)$. Then G can be decomposed into two connected subgraphs, H_1, H_2 , that are connected by a 2-path P in G . We will show that in this case $\mathcal{I}$ can be solved in FPT-time.

Call a vertex $x \in V(G)$ *critical* if $\text{dist}_G(x, w) \leq k$ for some $w \in V(H_1) \cup V(H_2) \cup \{s_i, t_i \mid R_i \in \mathcal{R}\}$. That is, x is critical if it is within distance at most k from a vertex in one of the

two subgraphs H_1, H_2 , or if it is within distance at most k from the starting position or the ending position of some robot.

Initially, all robots are located at critical vertices (since their starting vertices are critical). It is not difficult to prove, by induction on the number of moves and by normalizing the moves in a solution, that there exists an optimal solution to $\mathcal{I}$ in which if a robot $R \in \mathcal{R}$ is not a critical vertex, then it is moving directly without stopping or changing direction on P between two critical vertices.

Define a *configuration* to be an injection from $\mathcal{R}$ into the set of critical vertices in G . Define the configuration graph $\mathcal{C}_G$ to be the weighted directed graph whose vertex-set is the set of all configurations, and whose edges are defined as follows. There is an edge from a configuration q to a configuration q' if and only if q' can be obtained from q via a single (parallel) move of a subset of the robots, or q and q' are identical with the exception of a single robot's transition from a critical vertex on P to another critical vertex on P along an unoccupied path; in either case, the weight of edge (q, q') is the total traveled length by all the robots for the corresponding move/transition.

To decide $\mathcal{I}$, we compute the shortest paths (e.g., using Dijkstra's algorithm) between the initial configuration, defined based on the input instance (in which every robot is at its initial position), and every configuration in the configuration graph in which the robots in $\mathcal{M}$ are located at their destinations, and then choose a shortest path, among the computed ones, with total length at most ℓ if any exists.

The number of critical vertices is at most $2k \cdot 2|\mathcal{R}| + |V(H_1)| + |V(H_2)| \leq 4k^2 + 16k$, and hence the number of configurations is at most $(4k^2 + 16k)^k$. It follows that the problem can be solved in $\mathcal{O}^*((4k^2 + 16k)^{2k})$ time, and hence is FPT. ◀

► **Lemma F.** *Let $\mathcal{I} = (G, \mathcal{R} = (\mathcal{M}, \mathcal{F}), k, \ell)$ be an instance of GCMP1, and let $\text{opt}(\mathcal{I})$ be an optimal solution for $\mathcal{I}$. Let v be a type-(3) vertex, and let Q be the connected subgraph of size at most $8k$ that is connected by a 2-path P to a nice vertex n_R , and such that $v \in V(Q) \cup V(P)$. If a robot $R \in \mathcal{F}$ starts on v , then the route of R in $\text{opt}(\mathcal{I})$ cannot visit n_R and then visit a vertex in $V(Q)$.*

Proof. Let $R_1 \neq R$ be the single robot in $\mathcal{M}$. Observe that the only reason for R to visit n_R is to reverse order with R_1 whose destination must be in this case in $V(P) \cup V(Q)$. (Otherwise, R would never have to leave $P \cup Q$.) But then R_1 must enter $P \cup Q$ for the last time before R does, and hence R will never end up in a vertex in Q in $\text{opt}(\mathcal{I})$ since $|P| \geq 3k$. ◀

► **Lemma G.** *Let $\mathcal{I} = (G, \mathcal{R} = (\mathcal{M}, \mathcal{F}), k, \ell)$ be an instance of GCMP1, and let $\text{opt}(\mathcal{I})$ be an optimal solution for $\mathcal{I}$. Let v be a type-(3) vertex, and let Q be the connected subgraph of size at most $8k$ that is connected by a 2-path P of length at least $k + 1$ to a nice vertex n_R , and such that $v \in V(Q) \cup V(P)$. If a robot $R \in \mathcal{F}$ starts on v , then the route of R in $\text{opt}(\mathcal{I})$ cannot visit a vertex in $V(Q)$ and then visit n_R .*

Proof. Let $R_1 \neq R$ be the single robot in $\mathcal{M}$. The robot R might visit Q because it is pushed inside by R_1 or by some other robot $R' \in \mathcal{F}$. If it is pushed by R_1 to change their order in Q , then it would never need to go as far as n_R (since $|P| \geq k + 1$) after visiting Q to make space for R_1 . On the other hand, if it is pushed by another robot R' , then since neither of the two has a destination, we can exchange R and $R' \in \mathcal{F}$, and change the schedule to a shorter one in which R' goes to n_R and R ends up in Q instead. ◀

► **Lemma H.** Let $\mathcal{I} = (G, \mathcal{R} = (\mathcal{M}, \mathcal{F}), k, \ell)$ be an instance of GCMP, v a nice vertex, H_v a haven for v , and S a schedule for $\mathcal{I}$. There is a polynomial time algorithm that takes S and computes a schedule S' such that:

1. each robot $R_i \in \mathcal{R}$ enters H_v at most once and leaves H_v at most once; moreover, if R_i starts in H_v , then it will never re-enter H_v if it leaves it;
2. the total traveled length of all the robots outside of H_v in S' is upper-bounded by the total traveled length of all the robots outside of H_v in S ; and
3. the total traveled length of all the robots inside of H_v in S' is $\mathcal{O}(k^4)$.

Proof. We will construct the schedule S' as follows. For every robot $R_i \in \mathcal{R}$, define v_{first}^i and v_{last}^i as the first and last vertex, respectively, that R_i visits in H_v . Let W_i be the route for R_i in S . We let its route W'_i in S' follow W_i until it reaches the vertex v_{first}^i for the first time, with a slight modification that we might “freeze” R_i at its place for several time-steps when some other robot is either about to enter H_v or is leaving H_v . In addition, when R_i enters H_v in v_{first}^i , we change its routing in a way that it stays in H_v until it is its time to leave from v_{last}^i and never returns to H_v again. That is, when a robot R_i is about to enter v_{first}^i , we “freeze” the execution of the schedule and reconfigure the robots currently in H_v so that v_{first}^i is empty, for example by using Lemma 10, and then move R_i to v_{first}^i . From now on, we keep R_i in H_v , possibly moving it around each time we “freeze” the execution of the schedule to let a robot enter H_v or leave H_v . When all the robots are either in H_v or on the positions where they are at the time when R_i moves from v_{last}^i to some neighbor of v_{last}^i and never returns to H_v afterwards, we “freeze” the execution of the schedule and we reconfigure H_v to a configuration where R_i is at v_{last}^i by using Lemma 10. From this position R_i again follows W_i . It is easy to see that the total traveled length of S' outside of H_v is at most the total traveled length of S outside of H_v , as each robot is taking exactly the same steps until it reaches H_v for the first time, then it stays in H_v and after it leaves H_v , it is repeating the same steps as it does in S after leaving H_v for the last time. In addition, all the robot movements inside of H_v took place only when we froze the execution of the schedule to either let a robot enter H_v or leave H_v . This happens at most $2k$ times and each reconfiguration takes $\mathcal{O}(k^3)$ steps by Lemma 10. Hence the total traveled length of S' inside of H_v is at most $\mathcal{O}(k^4)$. Since we also modified the solution such that every robot enters and leaves H_v in S' at most once, the lemma follows.³ ◀

► **Lemma I.** $\mathcal{I} = (G, \mathcal{R} = (\mathcal{M}, \mathcal{F}), k, \ell)$ be an instance of GCMP. Let v be a nice vertex and H_v be its haven. Then in any optimal solution $opt(\mathcal{I})$ for $\mathcal{I}$, the total traveled length of all robots in $\mathcal{R}$ in H_v , as well as the total number of vertices of H_v visited by $\mathcal{R}$, is $\mathcal{O}(k^4)$.

Proof. Assume, for contradiction, that the total length traveled of $opt(\mathcal{I})$ inside H_v is at least $c \cdot k^4 + 2k + 1$, where c is the constant in big-Oh notation of Lemma H. Then by Lemma H, we can turn $opt(\mathcal{I})$ into a schedule S such that each robot moves exactly the same outside of H_v in both S and $opt(\mathcal{I})$, each robot passes an edge between H_v and the rest of the graph at most twice, and the total length traveled by all the robots inside H_v in S is at most $c \cdot k^4$. But then the total length traveled in S is smaller than the total length traveled by $opt(\mathcal{I})$, which is a contradiction. ◀

³ With a more careful analysis, it is possible to show that we only need at most k steps to enter H_v and at most $\mathcal{O}(k^2)$ to leave.

► **Lemma J.** $\mathcal{I} = (G, \mathcal{R} = (\mathcal{M}, \mathcal{F}), k, \ell)$ be an instance of GCMP. Then in any optimal solution $\text{opt}(\mathcal{I})$, the total travelled length of any $R \in \mathcal{F}$ that is at distance λ from a nice vertex is $\mathcal{O}(\lambda \cdot k + k^4)$.

Proof. Assume for a contradiction that $R \in \mathcal{F}$ starts at a vertex $s_R \in V(G)$ that is at distance λ from a nice vertex n_R , but the traveled length of R in $\text{opt}(\mathcal{I})$ is at least $2\lambda \cdot k + c_1 \cdot k^3 + c_2 \cdot k^4 + 2k + 1$, where c_1 is the constant in the big-Oh notation from Lemma 10 and c_2 is the constant in big-Oh notation from Lemma H.

We can now modify $\text{opt}(\mathcal{I})$ by first moving R inside H_v , this can be done with total traveled length at most $2\lambda \cdot k + c_1 \cdot k^3$ by pushing all robots along a shortest $s_R - n_R$ path to H_v using at most $\lambda \cdot k$ steps, reconfiguring inside of H_v using $c_1 \cdot k^3$ steps such that the robots that we pushed in H_v can leave back towards their starting positions using additional $\lambda \cdot k$ steps. Afterwards, we follow $\text{opt}(\mathcal{I})$, keeping R always in H_v and whenever a different robot interacts with H_v , we reconfigure it using Lemma 10 to the position from which it needs to leave next. Let this schedule be S . Note that the total traveled length of S outside of H_v is at most $2\lambda \cdot k + c_1 \cdot k^3$ larger than the total traveled length of $\text{opt}(\mathcal{I})$ outside of H_v and R stays in H_v after it entered it for the first time. By Lemma H, we can adapt S to a schedule S' such that S' still does only at most $2\lambda \cdot k + c_1 \cdot k^3$ more steps outside of H_v than $\text{opt}(\mathcal{I})$ does, but does at most $2k$ steps between H_v and $G - H_v$ and at most $c_2 \cdot k^4$ steps inside of H_v , which contradicts the optimality. ◀

Before we are ready to prove the main theorem for this section, we first need the following lemma, which will allow us to restrict the movement of the free robots.

► **Lemma 14.** Let $\mathcal{I} = (G, \mathcal{R} = (\mathcal{M}, \mathcal{F}), k, \ell)$ be an instance of GCMP. For every $R_i \in \mathcal{F}$ that is within distance λ_i from a nice vertex, there is a set $B(R_i, \lambda_i)$ of vertices of cardinality $k^{\mathcal{O}(\lambda_i \cdot k + k^4)}$ such that, for any optimal solution $\text{opt}(\mathcal{I})$, there exists an optimal solution $\text{opt}'(\mathcal{I})$ satisfying that, in $\text{opt}'(\mathcal{I})$ R_i moves only on vertices in $B(R_i, \lambda_i)$ and all the remaining robots move exactly the same in $\text{opt}(\mathcal{I})$ and $\text{opt}'(\mathcal{I})$. Moreover, $B(R_i, \lambda_i)$ is computable in $\mathcal{O}(|V(G)| + |E(G)|)$ time.

Proof. First observe that if a vertex v has degree at least $c_1 \cdot k^4 + k + 1$, where c_1 is the constant from Lemma I, then any connected subgraph H_v of G that consists of v and $k^4 + k + 1$ many arbitrary neighbors of v is a haven. By Lemma I, any optimal solution visits at most $c_1 \cdot k^4 + k$ vertices of H_v and hence also at most $c_1 \cdot k^4 + k$ neighbors of v . This is because every robot enters H_v at most once, and the total traveled length inside H_v is at most $c_1 \cdot k^4$. It follows that in any optimal schedule $\text{opt}(\mathcal{I})$, any free robot R_i that enters v does at most one step after entering v . Otherwise, we could get a shorter schedule by moving R_i , after entering v for the first time, directly to any neighbor of v in $V(H_v)$ that is not visited by any robot. In addition, if $\text{opt}(\mathcal{I})$ moves R_i outside H_v after visiting v , then we can modify the schedule $\text{opt}(\mathcal{I})$ to obtain schedule $\text{opt}'(\mathcal{I})$ in which R_i enters v and then moves to a neighbor w of v such that $w \in V(H_v)$ and w is not visited by any other robot in $\text{opt}(\mathcal{I})$. By Lemma J, the total traveled length of any free robot $R_i \in \mathcal{F}$ is at most $c_2(\lambda_i \cdot k + k^4)$. From the discussion above, to obtain $B(R_i, \lambda_i)$, we only need to keep vertices that can be reached from the starting position s_i of R_i by a path P of length at most $c_2(\lambda_i \cdot k + k^4)$ such that only the penultimate vertex on P can have degree larger than $c_1 \cdot k^4 + k$ in G . Moreover, for every vertex that has degree larger than $c_1 \cdot k^4 + k$, we only need to keep $c_1 \cdot k^4 + k$ arbitrary neighbors. It follows that we can obtain the set $B(R_i, \lambda_i)$ by first a modified depth-bounded BFS from s_i , such that for any vertex v of degree at least $c_1 \cdot k^4 + k + 1$, we do not continue the BFS from this vertex. The depth of this BFS

is bounded by $c_2(\lambda_i \cdot k + k^4)$ and it is easy to see that the number of vertices marked by this BFS to be included in $B(R_i, \lambda_i)$ is at most $(c_1 \cdot k^4 + k + 1)^{1+c_2(\lambda_i \cdot k + k^4)}$. Afterwards, for every vertex in $B(R_i, \lambda_i)$ with degree at least $c_1 \cdot k^4 + k + 1$, but with less than $c_1 \cdot k^4 + k + 1$ many neighbors in $B(R_i, \lambda_i)$, we add arbitrary $c_1 \cdot k^4 + k + 1 - |N_G(v) \cap B(R_i, \lambda_i)|$ many neighbors of v to $B(R_i, \lambda_i)$. This concludes the construction of $B(R_i, \lambda_i)$. It is easy to see that $|B(R_i, \lambda_i)| \leq (c_1 \cdot k^4 + k + 1)^{1+c_2(\lambda_i \cdot k + k^4)} \cdot (c_1 \cdot k^4 + k + 1) = k^{\mathcal{O}(\lambda_i + k^4)}$ and the lemma follows. $\blacktriangleleft$

► **Theorem 15.** *Let $\mathcal{I} = (G, \mathcal{R} = (\mathcal{M}, \mathcal{F}), k, \ell)$ be an instance of GCMP. In FPT-time, we can reduce $\mathcal{I}$ to $p = g(k)$ -many instances (for some computable function $g(k)$) $\mathcal{I}_1, \dots, \mathcal{I}_p$ such that there exists $j \in [p]$ satisfying:*

1. *If $\mathcal{M} = \{R_1\}$ then $\mathcal{I}_j = (G_j, \mathcal{R}_j = (\mathcal{M}_j, \emptyset), k_j, \ell_j)$, where $k_j \leq k$ and $\ell_j \leq \ell$, is a yes-instance of GCMP if and only if $\mathcal{I}$ is a yes-instance. Moreover, in this case there is an optimal solution $\text{opt}(\mathcal{I}_j)$ such that $|\text{opt}(\mathcal{I}_j)| = \text{dist}_{G_j}(s_1, t_1) + \mathcal{O}(k^5)$, and for every robot $R_i \in \mathcal{M}_j \setminus \{R_1\}$, the moves of R_i in $\text{opt}(\mathcal{I}_j)$ are restricted to a vertex set of cardinality at most $k^{\mathcal{O}(k^4)}$ that is computable in linear time.*
2. *In polynomial time we can compute a schedule for $\mathcal{I}_j$ with total traveled length at most $\sum_{R_i \in \mathcal{M}_j} \text{dist}(s_i, t_i) + \mathcal{O}(k^5)$, and given such a schedule for $\mathcal{I}_j$, in polynomial time we can compute a schedule for $\mathcal{I}$ of cost at most $|\text{opt}(\mathcal{I})| + \mathcal{O}(k^5)$.*

Furthermore, every optimal schedule for $\mathcal{I}$ satisfies the property that every vertex in G is visited at most $\mathcal{O}(k^5)$ many times in the schedule.

Proof. Let $\mathcal{I} = (G, \mathcal{R} = (\mathcal{M}, \mathcal{F}), k, \ell)$ be an instance of the problem, and let $\text{opt}(\mathcal{I})$ denote an optimal schedule for $\mathcal{I}$. The reduction will produce a set of instances $\mathcal{I}_1, \dots, \mathcal{I}_p$, where $p = g(k)$, such that one of the instances $\mathcal{I}_j$, $j \in [p]$, satisfies the statement of the theorem. For ease of presentation, we will present a nondeterministic reduction that produces a single instance $\mathcal{I}_j = (G_j, \mathcal{R}_j = (\mathcal{M}_j, \emptyset), k_j, \ell_j)$, and the reduction can be made deterministic in FPT-time to produce a set of $[p]$ instance, one of them, $\mathcal{I}_j$, satisfies the statement of the theorem. We will assume that the reduction makes the right guesses; when we simulate the reduction deterministically, we will discard any enumeration resulting in inconsistent or incorrect guesses. To construct $\mathcal{I}_j$, we will be making guesses about the initial and final segments of the robots' routes in $\text{opt}(\mathcal{I})$, which may lead to redefining the starting and final positions for some of them. The guessing is based on the types of the starting and final positions of the robots (see Lemma 11). Set $\ell_j = \ell$. Let R be a robot located at a vertex v_R . If v_R is nice then we keep R at v_R .

If v_R is a type-(1) vertex, then we do nothing.

If v_R is a type-(2) vertex, then v_R is on a 2-path P whose both endpoints are nice vertices, n_R and n'_R . (For visualization purposes, think about n'_R as being the “left” endpoint of P and n_R being its “right” endpoint.) Note that $|P| > 6k$ (otherwise, R would be within distance $3k + 1$ from a nice vertex and v_R would be of type-(1)). Let S be the set of robots that occupy vertices on P . We guess a partition of S into three subsets $S_{\text{Left}}, S_{\text{Mid}}, S_{\text{Right}}$, where S_{Left} is the subset of robots such that the first nice vertex they visit is n'_R , S_{Right} is the subset of robots such that the first nice vertex they visit is n_R , and S_{Mid} is the set of robots that do not visit any nice vertex, and hence remain on P . Note that $S_{\text{Left}}, S_{\text{Mid}}, S_{\text{Right}}$ may contain robots in $\mathcal{F}$. Observe that S_{Left} must be the $|S_{\text{Left}}|$ -many closest robots in S to n'_R and S_{Right} must be the closest $|S_{\text{Right}}|$ -many robots in S to n_R . We route the robots in S_{Left} and S_{Right} as follows. For each robot $R' \in S_{\text{Left}}$ (resp. S_{Right}), move it towards n'_R (resp. n_R), pushing along robots on its way if needed, until the first time it reaches a vertex that is at distance at most k from n'_R (resp. n_R). Note that each robot in $S_{\text{Left}} \cup S_{\text{Right}}$

is now within distance k from a nice vertex; let ℓ_1 be the total traveled length during this routing. We change the starting position of each robot in $S_{\text{Left}} \cup S_{\text{Right}}$ in the instance $\mathcal{I}$ to its current position after the routing and subtract ℓ_1 from ℓ_j .

If v_R is a type-(3) vertex, let C be the connected subgraph of size at most $8k$ that is connected by a 2-path P to a nice vertex n_R , and such that $v_R \in V(C) \cup V(P)$. If $|P| \leq 3k$ then we do nothing, as in this case v_R will be treated later as a type-(1) vertex. Let u_R be the vertex connecting P to C .

Suppose now that $|P| > 3k$, let S be the subset of robots occupying vertices in $V(P)$ and let S_C be the set of robots occupying vertices in $V(C)$. We treat this case similarly to how we treated type-(2) vertices with respect to n_R and u_R , and guess a partitioning of S into three subsets $S_{\text{Left}}, S_{\text{Mid}}, S_{\text{Right}}$, where S_{Left} is the subset of robots such that the first vertex with degree at least three in G they visit is n_R , S_{Right} is the subset of robots such that the first vertex with degree at least three in G they visit is u_R , and S_{Mid} is the subset of robots that do not visit any nice vertex nor visit u_R , and hence never leave P . We route the robots in S_{Left} and S_{Right} as follows. For each robot $R' \in S_{\text{Left}}$ (resp. S_{Right}), move it towards n_R (resp. u_R), pushing along robots on its way if needed, until the first time it reaches a vertex that is at distance at most k from n_R (resp. u_R). Let ℓ_2 be the total traveled length during this routing. We redefine the starting position of each robot in $S_{\text{Left}} \cup S_{\text{Right}}$ in the instance $\mathcal{I}_j$ to be its current position after the routing and subtract ℓ_2 from ℓ_j . Now if $|\mathcal{M}| = 1$, then by Lemmas F and G, no free robot starts in P and either visits n_R and then comes back to C , or visits C and then visits n_R . We note that in the case that R_1 either starts or ends in C , it is possible that free robots in C cannot make way for R_1 to leave/enter C and need to leave through n_R . It is not difficult to see that in this case $S_{\text{Right}} \cup S_{\text{Mid}} = \emptyset$. We can guess which robots in C need to enter n_R and use Lemma 13 to compute a schedule with the minimum total traveled length that lets precisely these robots leave C toward n_R . Afterwards, we compute the routing of the guessed robots that takes them to vertices in B_{Left} and subtract the total traveled length of this routing from ℓ_j . We are done with this case for now. We note that, assuming that the guesses are correct, in all the above treatments, we maintain the equivalence between the two instances $\mathcal{I}$ and $\mathcal{I}_j$. The following treatment for the case where $|\mathcal{M}| > 1$ may result in a non-equivalent instance as the treatment may result in an additive error of $\mathcal{O}(k^4)$ steps.

If $|\mathcal{M}| > 1$, we guess which of the robots in $S_{\text{Right}} \cup S_C$ visit n_R after visiting C and guess the order in which they leave C for the last time before they visit n_R for their first time. We know that the optimal solution does route these robots to leave towards n_R in this order, possibly with some additional robots entering C . Hence, it must be possible to reorder these robots as in their guessed order. Note that during this reordering, the remaining robots in $S_C \cup S_{\text{Right}}$ might need to move to some positions in C or on the closest k vertices to C . We guess the exact positions where they end after this reordering, and let these guessed positions be their new starting positions. By Proposition 8, in $\mathcal{O}(k^3)$ steps, we can reconfigure the robots that leave through n_R in the correct order on P and the remaining robots in $S_C \cup S_{\text{Right}}$ to their new starting positions. Afterwards, we can move these robots to S_{Left} and shift their starting positions to be at distance at most k from n_R as we did above for the robots that started in S_{Left} . Let ℓ_3 be the total traveled length during this rerouting. We subtract ℓ_3 from ℓ_j .

For a robot $R \in \mathcal{M}$, let t_R be its destination.

If t_R is within distance $11k$ from a nice vertex (which includes t_R being a type-(1) vertex), then we distinguish three cases depending on whether the current starting position s'_R of R is also at distance at most $11k$ from a nice vertex, or s'_R is of type-(2), or s'_R is of type-(3).

If s'_R is at distance at most $11k$ from a nice vertex, we do nothing. If s'_R is of type-(2), since it is not close to a nice vertex, then our guess for R was that it would never leave the path P associated with this type-(2) vertex. We treat this case as we do below as if t_R were a type-(2) vertex. If s'_R is of type-(3), since it is not close to a nice vertex, then our guess for R was that it would never leave $P \cup C$, where P and C , respectively, are the 2-path and the connected subgraph associated with this type-(3) vertex. We treat this case as we do below as if t_R were a type-(3) vertex.

If t_R is a type-(2) vertex, which is on a 2-path P whose endpoints n_R and n'_R are nice, then for each $R' \in \mathcal{F}$, guess whether it ends on P and guess its relative position on P with respect to all the robots in $\mathcal{R}$ that end on P . Note that such a robot R' either visits a nice vertex in $\text{opt}(\mathcal{I})$, in which case its starting position is at distance at most $11k$ from a nice vertex, in which case we can assume that it ends on a vertex in $B(R', 11k)$ (see Lemma 14) and only allow the guess that R' ends on P if P intersects $B(R', 11k)$. Alternatively, if R' starts on P and never leaves it, it will be either later removed from $\mathcal{I}_j$, or restricted to k vertices on P that are closest to either n'_R or n_R . In either case, we can restrict the moves of R' to a set $B(R')$ of vertices satisfying $|B(R')| = k^{\mathcal{O}(k^4)}$.

Now let T be the set of robots that end on vertices on P . We guess a partitioning of T into three subsets $T_{\text{Left}}, T_{\text{Mid}}, T_{\text{Right}}$, where T_{Left} is the subset of robots such that the last nice vertex they visit is n'_R , T_{Right} is the subset of robots such that the last nice vertex they visit is n_R , and T_{Mid} is the subset of robots that do not visit any nice vertex, and hence never leave P . Note that T_{Mid} has to be equal to the set S_{Mid} that was guessed above while considering the types of the starting positions of the robots w.r.t. P .

Let B_{Left} and B_{Right} be the sequences of the k closest vertices on P_R to n'_R and to n_R , respectively. For each robot $R_i \in T_{\text{Left}}$, let x_i be the number of robots in $T_{\text{Left}} \cup T_{\text{Mid}}$ whose relative position on P is farther from n'_R than that of R_i . If R_i has a destination t_i , we define a new destination t'_i for R_i to be the vertex on P whose distance from n'_R is equal to $\min\{\text{dist}_G(n'_R, t_i), k - x_i\}$. On the other hand, if $R_i \in \mathcal{F}$, then we guess t'_i to be any position at distance at most $k - x_i$ from n'_R that conforms to the guessed relative position of R_i with respect to the other robots on P . We do the same for the robots in T_{Right} .

For robots in T_{Left} , we now compute the routing that takes each robot R_i from t'_i to t_i directly, if needed pushing along the (free) robots (note that this might include free robots in T_{Mid}) on its way. We let ℓ_3 be the total traveled length during this computed routing and we subtract ℓ_3 from ℓ_j .

If $T_{\text{Mid}} \neq \emptyset$, let $R_i \in T_{\text{Mid}} \cap \mathcal{M}$ be a robot with starting position s_i and destination t_i . If $\{s_i, t_i\} \subseteq B_{\text{Left}}$, then we do nothing with R_i . Notice that all robots in S_{Left} must have their starting positions closer to n'_R than s_i and all robots in T_{Left} must have their destinations closer to n'_R than t_i ; therefore, none of the changes that we did to the starting positions and the destinations of robots in $S_{\text{Left}} \cup T_{\text{Left}}$ affects R_i . We do the same if $\{s_i, t_i\} \subseteq B_{\text{Right}}$. For the remaining robots in $T_{\text{Mid}} \cap \mathcal{M}$, we compute the routing that takes them directly from their starting positions to their destinations, possibly pushing along free robots in T_{Mid} . We let the total distance traveled by all these robots in T_{Mid} during this routing be ℓ_4 , we subtract ℓ_4 from ℓ_j , and we then remove these robots, as well as all the robots they pushed, from the instance. It is not difficult to see that in $\text{opt}(\mathcal{I})$ the travel length incurred by these robots is precisely ℓ_4 . This is because the robots in T_{Mid} have to be in a sense ordered on P . In addition, if a robot R' uses P only to avoid other robots passing through a nice vertex n'_R (resp. n_R), after R' enters P from n'_R (resp. n_R), if R leaves P through the same nice vertex n'_R (resp. n_R), then R' will remain in B_{Left} (resp. B_{Right}). Therefore, we can start by moving any of these robots in T_{Mid} that start away from B_{Left} and B_{Right} towards their destinations,

possibly pushing some robots just outside of these buffers closer to their destinations. Then we can follow an optimal solution to move the robots in T_{Left} and T_{Right} to their destinations in $\mathcal{I}_j$ when all the robots on P are in the correct final relative order, and lastly just navigate each robot directly to its destination.

Observe now that what we are left to deal with are the robots in $\mathcal{F} \cap T_{\text{Mid}}$, which appear either as the first or last sequence of robots in T_{Mid} w.r.t. P . Note that, if there is such a robot R_i whose s_i is not in $B_{\text{Left}} \cup B_{\text{Right}}$, then we do not need to do anything since any traveled length incurred by R_i has already been accounted for in the previous cases, and we can remove R_i from the instance. Suppose now that s_i is in one of the two buffers, say B_{Left} . If there is a robot in T_{Left} that pushes R_i outside of B_{Left} , then when we computed its final routing contribution to ℓ_3 , the travel length incurred by R_i has been already accounted for. So now we can assume that the final position of R_i is in B_{Left} . We guess the final position of t_i of R_i in B_{Left} . Note that the robots that we kept in $\mathcal{I}_j$ have their starting positions and their destinations in $B_{\text{Left}} \cup B_{\text{Right}}$. If T_{Mid} was non-empty at the beginning, this means that P cannot be used by any robots to go between n'_R and n_R , and in this case we remove $V(P) \setminus (B_{\text{Left}} \cup B_{\text{Right}})$ from the graph.

If t_R is a type-(3) vertex, let C be the connected subgraph of size at most $8k$ that is connected by a 2-path P to a nice vertex n_R , and such that $v_R \in V(C) \cup V(P)$. If $|P| \leq 3k$ then we do nothing, as in this case t_R will be treated later as a type-(1) vertex. Let u_R be the vertex connecting P to C .

Similarly to how we treated the type-(2) vertices above, for each $R' \in \mathcal{F}$, we guess whether it ends on P and guess its relative position on P with respect to all the robots in $\mathcal{R}$ that end on P . If a robot in $\mathcal{F}$ does not end on P , we guess whether it ends in C , and in which case we also guess the exact vertex in C on which the robot ends. Again, as in the above treatment of type-(2) vertices, R' either visits a nice vertex in $\text{opt}(\mathcal{I})$, in which case its starting position is at distance at most $11k$ from a nice vertex in $\mathcal{I}_j$, and in this case we can assume that it ends on a vertex in $B(R', 11k)$ (see Lemma 14), and only allow the guess that R' ends on $C \cup P$ if $V(P \cup C)$ intersects $B(R', 11k)$. Alternatively, if R' starts on $P \cup C$ and never leaves it, in which case it will be either later removed from $\mathcal{I}_j$, restricted to k vertices on P that are closest to n_R , or restricted to C and the k vertices on P closest to u_R . In either case, we can restrict the moves of R' in an optimal solution to a set $B(R')$ of vertices, where $|B(R')| = k^{\mathcal{O}(k^4)}$.

Now let T be the set of robots that end on P and T_C be the set of robots that end in C . We guess a partitioning of T into three subsets $T_{\text{Left}}, T_{\text{Mid}}, T_{\text{Right}}$, where T_{Left} is the subset of robots such that the last degree three vertex they visit is n_R , T_{Right} is the subset of robots such that the last degree three vertex they visit is u_R , and T_{Mid} is the subset of robots that do not visit any nice vertex, and hence never leave P . Note that T_{Mid} has to be the same as the set S_{Mid} that was guessed above while considering the types of the starting positions of the robots w.r.t. P .

Let B_{Left} be the sequence of the k closest vertices on P to n_R . For each robot $R_i \in T_{\text{Left}}$, let x_i be the sum of the number of robots in T_C and the number of robots in T whose relative destination on P is farther from n_R than that of R_i . If R_i has destination t_i , we define a new destination t'_i for R_i to be the vertex in B_{Left} whose distance from n_R is equal to $\min\{\text{dist}_G(n'_R, t_i), k - x_i\}$. On the other hand, if R_i is in $\mathcal{F}$, then we guess a destination t'_i for R_i from among the positions at distance at most $k - x_i$ from n_R conforming to the guessed relative destinations with respect to the other robots on P .

For the robots in T_{Left} , we now compute the routing that takes each robot R_i from t'_i to t_i directly, if needed pushing along the (free) robots (note that this might include free robots

in T_{Mid}) on its way. We let ℓ_5 be the total traveled length during this computed routing and we subtract ℓ_5 from ℓ_j .

If $T_{\text{Mid}} \neq \emptyset$, then we observe that T_{Mid} separates the robots in $T_{\text{Mid}} \cup T_{\text{Right}} \cup T_C$ from the rest of the instance. This means that the remaining robots cannot use C to change their order on P , and since their destinations are either in $V(G) \setminus (V(P) \cup V(C))$ or in B_{Left} , they never need to enter vertices of $V(P) \setminus B_{\text{Left}}$. (Note that in the modified instance, the destinations of the robots in T_{Left} have been changed to vertices in B_{Left} .) If a robot in T_{Mid} starts and ends in B_{Left} , it still might need to be pushed around by other robots in the solution; let T'_{Mid} be the set of robots in $T_{\text{Mid}} \cup T_{\text{Right}} \cup T_C$ except those in T_{Mid} that start and end in B_{Left} . We now use Lemma 13 to compute an optimal routing for T'_{Mid} in $G[V(P) \cup V(C)]$. Note that since P has length at least $3k$, there is an optimal solution that starts by moving the robots that start in B_{Left} outside of B_{Left} , routes the robots so that the robots whose destinations in C are on their correct destinations, and such that the robots with destinations on P are positioned on vertices in $V(P) \setminus B_{\text{Left}}$ in the correct relative position without any robot ever entering B_{Left} and finishes by routing the robots on P directly to their destinations. Let ℓ_6 be the total traveled length during this optimal routing. We subtract ℓ_6 from ℓ_j , remove the robots in T'_{Mid} from the instance and the vertices in $V(C) \cup (V(P) \setminus B_{\text{Left}})$ from the graph.

Now, if $T_{\text{Mid}} = \emptyset$, we treat T_{Right} similarly to T_{Left} . That is, we define B_{Right} to be the sequence of the k closest vertices on P to u_R . For a robot $R_i \in T_{\text{Right}}$, we let x_i be the sum of the number of robots in T_C and the number of robots in T whose relative destinations on P are farther from n_R than that of R_i (i.e., closer to u_R).

If R_i has destination t_i , we define a new destination t'_i for R_i to be the vertex in B_{Right} whose distance from u_R is equal to $\min\{\text{dist}_G(u_R, t_i), x_i + 1\}$. On the other hand, if R_i is in $\mathcal{F}$, then we guess a destination t'_i for R_i from among the positions at distance at most $x_i + 1$ from u_R conforming to the guessed relative destination of R_i with respect to the other robots on P . We now compute the routing that takes each robot $R_i \in T_{\text{Right}}$ from t'_i to t_i directly, if needed pushing along the (free) robots (note that this might include free robots in T_{Mid}) on its way. We let ℓ_7 be the total traveled length during this computed routing and we subtract ℓ_7 from ℓ_j .

Now if $|\mathcal{M}| = 1$, then by Lemmas F and G, no robot starts in $P \cup C$ and either visits n_R and then comes back to C , or visits C and then visits n_R , and we are done with this case. We recall that, assuming that the guesses are correct, in all the above treatments for the destination vertices, we maintain the equivalence between the two instances $\mathcal{I}$ and $\mathcal{I}_j$. The following treatment for the case where $|\mathcal{M}| > 1$ may result in non-equivalent instance, as the treatment may result in an additive error of $\mathcal{O}(k^4)$ steps.

If $|\mathcal{M}| > 1$, we guess which of the robots in $T_{\text{Right}} \cup T_C$ visit n_R before visiting C and guess the order in which they leave n_R for the last time before they visit C ; denote the set of these robots by $T_{n_R}^C$. For every such robot R_i , we now define a new destination t''_i in B_{Left} in the same manner as we did for the robots in T_{Left} and move R_i to T_{Left} (and remove it from $T_{\text{Right}} \cup T_C$). We know that the optimal solution does route each robot R_i from t''_i to t'_i . To upper bound the contribution of such reconfiguration to the total traveled length, we observe that after all of these robots are at their positions t''_i and only the remaining robots in $T_{\text{Right}} \cup T_C$ are in $V(C) \cup (V(P) \setminus B_{\text{Left}})$, then the robot $(T_{\text{Right}} \cup T_C) \setminus T_{n_R}^C$ are in $V(C) \cup B_{\text{Right}}$ (they will not need to move since we moved them when considering the starting positions in $V(C) \cup V(P)$), and we can route the robots in $T_{n_R}^C$ first to the buffer B_{Right} by going directly there, maybe pushing some of the robots in B_{Right} closer to u_R . Afterwards, we use Proposition 8 to move the robots in the correct order in B_{Right} in $\mathcal{O}(k^3)$ steps. Let ℓ_8

be the total traveled length during this reconfiguration. We subtract ℓ_8 from ℓ_j and remove the robots left in $T_C \cup T_{\text{Right}}$ from the instance and the vertices in $V(C) \cup (V(P) \setminus B_{\text{Left}})$ from the graph.

Now, we know that the optimal solution does route these robots to leave towards C in this order possibly even interacting with some robots that started in C , but do not end there. Moreover, we can use the algorithm of Lemma 13 to reconfigure these robots in the correct order on P . Afterwards, we can move these robots to S_{Right} and shift their starting positions to B_{Right} in the same manner as we did above for the robots that started in S_{Right} . Let ℓ_9 be the total traveled length during this reconfiguration. We subtract ℓ_9 from ℓ_j .

Observe that some of the free robots still do not have destinations, but may have remained in the instance. According to our guesses, each such robot R is either within a distance at most $11k$ from some nice vertex, or starts in a type-(3) vertex such that the 2-path P associated with that type-(3) vertex has length at least $3k$, and never leaves $C \cup B_{\text{Right}}$, where C is the connected subgraph of size at most $8k$ associated with that type-(3) vertex and B_{Right} is the buffer of k vertices on P that are closest to C . By Lemma 14, we can assume that in $\text{opt}(\mathcal{I})$ the moves of every free robot R that is at distance at most $11k$ from a nice vertex are restricted to $B(R, 11k)$. Moreover, $B(R, 11k)$ has cardinality $k^{O(k^4)}$ and is computable in $\mathcal{O}(|V(G)| + |E(G)|)$ time. We can guess the vertex in $B(R, 11k)$ that R ends up on in $\text{opt}(\mathcal{I})$ and assign it as its final destination in $\mathcal{I}_j$. Define $B(R) = B(R, 11k)$ in this case. Now for each free robot R that is in $V(C) \cup B_{\text{Right}}$ and never leaves it, we guess which vertex, out of the at most $9k$ vertices in $V(C) \cup B_{\text{Right}}$ it ends up on in the optimal solution guessed above, and assign it as its final destination in $\mathcal{I}_j$. Define $B(R) = V(C) \cup B_{\text{Right}}$ in this case.

Let $\mathcal{I}_j = (G_j, \mathcal{R}_j = (\mathcal{M}_j, \emptyset), k_j, \ell_j)$ be the resulting instance from the above. It is clear that $k_j \leq k$ since we never added any robots to the instance. Moreover, in the case where $|\mathcal{M}| = 1$, assuming the guesses are correct, the equivalence between the two instances $\mathcal{I}$ and $\mathcal{I}_j$ follows (as argued above). Now to show in this case that for any optimal solution $\text{opt}(\mathcal{I}_j)$ of $\mathcal{I}_j$, we have $|\text{opt}(\mathcal{I}_j)| = \text{dist}_{G_j}(s_1, t_1) + \mathcal{O}(k^5)$, where R_1 is the only robot in $\mathcal{M}$, let P_1 be a shortest path in G_j between s_1 and t_1 , and note that $|P_1| = \text{dist}_{G_j}(s_1, t_1)$. Observe that every robot is now positioned at distance at most $11k$ from a nice vertex in G_j , or it is at a distance at most k from some connected subgraph C of a type-(3) vertex and will never visit the nice vertex associated with that type-(3) vertex. For every nice vertex w , let H_w be the haven associated with w as in Definition 9. First, we move all the robots within distance $11k$ from w into H_w .

Observe that P_1 can only interact with the connected subgraph C of a type-(3) vertex either at the beginning of P_1 before visiting any havens, or at the end when it will no longer visit a haven afterwards. This is true since P_1 is a (simple) shortest path and each such connected subgraph C is separated by a nice vertex from the rest of the graph. Suppose s_1 is at distance at most k from such a connected subgraph C separated by a nice vertex n_R ; the argument is similar in the case where t_1 is at distance at most k from C . Note that all the robots that are at a distance at most k from C are the robots that visit C before visiting n_R . Moreover, in an optimal solution, no such robot (other than R_1) visits n_R , as stipulated by the guess and Lemma G. Hence, the optimal solution does reconfigure these robots in such a way that R_1 is the closest robot to n_R among all of them, or reconfigure R_1 to its destination in C . In either of these cases, by Proposition 8, we can perform this reconfiguration using at most $\mathcal{O}(k^3)$ steps. After this reconfiguration (and similar one at the end of the routing), all the robots on P_1 are in havens.

Let us now iteratively modify P_1 to a final routing of R_1 in which R_1 enters and leaves

each haven at most once. Note that there are at most k havens that contain at least one robot and we repeat the following process at most k times each increasing the length of the routing by at most $\mathcal{O}(k^3)$. Let H_w be the first haven in which R_1 collides with a robot on P_1 . Let v_{first} be the first vertex in H_w that R_1 visits on P_1 and let v_{last} be the last vertex in H_w it visits on P_1 . We first reconfigure the robots on H_w such that v_{first} is empty. This can be done using at most $k - 1$ steps (at most $k - 1$ robots each doing one step). Now we follow P_1 until it enters v_{first} . Afterwards, we use Lemma 10 to reconfigure the set of all robots that are currently in H_w to obtain a resulting configuration in which R_1 is at v_{last} . The cost of this reconfiguration is $\mathcal{O}(k^3)$, and since this process is repeated at most k times, the overall additional cost incurred is $\mathcal{O}(k^4)$.

Note that technically in $\mathcal{I}_j$, $\mathcal{M}_j$ can contain other robots that R_1 and all these robots have destinations. However, all the robots that we did not navigate to the correct destinations are at the moment in havens, so the additional loss to the optimal solution that we did not account for to navigate R_1 from its destination in $\mathcal{I}_j$ (which is close to the haven) to its destination in $\mathcal{I}$ cannot be more than $\mathcal{O}(k^3)$, as it interacts in this case with at most one haven. Hence, we can assume that each of these robots is at distance at most $\mathcal{O}(k^3)$ from its destination, and we can iteratively navigate them to their destinations (which are close to havens), each incurring at most $\mathcal{O}(k^4)$ reconfiguration steps.

It follows that there is a schedule for R_1 of cost $\text{dist}_{G_j}(s_1, t_1) + \mathcal{O}(k^5)$, and the first part of Statement 1 of the theorem holds. Moreover, every robot in $\mathcal{R}_j$ other than R_1 has been assigned a final destination and its movement has been restricted to a set of vertices $B(R)$ of cardinality at most $k^{\mathcal{O}(k^4)}$, and the second part of Statement 1 of the theorem holds as well.

Let us now prove Statement 2 of the theorem, which states that in polynomial time we can compute a schedule for $\mathcal{I}_j$ with total traveled length at most $f(k) + \sum_{R_i \in \mathcal{M}_j} \text{dist}(s_i, t_i)$, and given such a schedule for $\mathcal{I}_j$, in polynomial time we can compute a schedule for $\mathcal{I}$ of cost at most $|\text{opt}(\mathcal{I})| + \delta(k)$, for some computable function δ .

First notice that all the modifications to the instance, except for those performed in the treatment of type-(3) vertices, preserve the optimality of the solution. In addition, in the treatment of type-(3) vertices, the additional possible cost over that of the optimal solution is $\mathcal{O}(k^4)$ to let all the robots first leave the connected subgraphs associated with type-(3) vertices and only after all the robots that wish to leave have left, the robots that need to enter can enter. This means that whenever we have a schedule for $\mathcal{I}_j$ of cost $\text{opt}(\mathcal{I}_j) + x$, for some $x \in \mathbb{N}$, we can get a solution for $\mathcal{I}$ of cost $\text{opt}(\mathcal{I}_j) + x + \mathcal{O}(k^4)$. This establishes the second part of Statement 2 of the theorem. Also observe that $\text{opt}(\mathcal{I}_j) \geq \sum_{R_i \in \mathcal{M}_j} \text{dist}(s_i, t_i)$. Note that thanks to this treatment of type-(3) vertices, all the starting positions and the destinations of the robots are at distance at most $11k$ from nice vertices. We start by pushing all the robots to the closest haven to their starting position. This takes at most $\mathcal{O}(k^2)$ many steps. We temporarily choose for each robot R_i an arbitrary destination t'_i in the haven that is closest to its final destination t_i . Note that this increases the distance of each robot to its destination by $\mathcal{O}(k)$ many steps. Now for each robot R_i , we compute a shortest path P_i from its current position in a haven to t'_i . We navigate each robot R_i along P_i to t'_i , treating the collisions in havens in exactly the same manner as we did for robot R_1 when proving Statement 1 of the theorem. This incurs a total cost of $\sum_{R_i \in \mathcal{M}_j} \text{dist}(s_i, t_i) + \mathcal{O}(k^5)$. For each haven H_w , compute a BFS tree rooted at w that spans the haven and the destinations of the robots that are in it. If two subtrees corresponding to havens H_u and H_w overlap, then we route all robots in H_u to H_w (note that the largest distance between two havens in a connected cluster is at most $22k^2$). This incurs a cost of $\mathcal{O}(k^3)$ steps overall. Afterwards, compute once again for the haven H_w a BFS tree rooted at w spanning each haven and the

destinations of the robots in it. Now we can reconfigure the robots so that they are on the path w to their destinations in the BFS tree and the robots on the same leaf-to-root path are in the same relative order as their destinations. This reconfiguration incurs a cost of $\mathcal{O}(k^4)$. Afterwards, each robot is at distance $\mathcal{O}(k^2)$ from its destination and we can send the robots directly to their destinations along the leaf-to-root path in the BFS tree. This incurs an additional $\mathcal{O}(k^3)$ many steps. Hence, the total traveled length by all robots is $\sum_{R_i \in \mathcal{M}_j} \text{dist}(s_i, t_i) + \mathcal{O}(k^5)$.

Finally, all the modifications performed while reducing $\mathcal{I}$ to $\mathcal{I}_j$, except for those $\mathcal{O}(k^4)$ steps needed in the treatment of type-(3) vertices, can be characterized as direct moves to each robot R_i (without going back-and-forth) between two vertices that R_i must visit in this order in $\text{opt}(\mathcal{I})$. Therefore, the optimal solution on which the guess for $\mathcal{I}_j$ is based, does take R_i from its starting position in $\mathcal{I}$ to its starting position in $\mathcal{I}_j$, then to its terminal position in $\mathcal{I}_j$ and finally from its terminal position in $\mathcal{I}_j$ to its destination in $\mathcal{I}$. If we just sum up these distances, and using the fact that optimal length for $\mathcal{I}_j$ is at most $\sum_{R_i \in \mathcal{M}_j} \text{dist}(s_i, t_i) + \mathcal{O}(k^5)$, we can conclude that $\text{opt}(\mathcal{I})$ can do only at most $\mathcal{O}(k^5)$ steps that are not moving R_i closer to either its start in $\mathcal{I}_j$, its destination in $\mathcal{I}_j$ or its destination in $\mathcal{I}$, and hence any vertex cannot be visited by a robot more than $\mathcal{O}(k^5)$ times. $\blacktriangleleft$

The approximation algorithm claimed in the introduction (restated below) follows directly from Statement 2 of Theorem 15.

► **Theorem 3.** *There is an FPT approximation algorithm for GCMP parameterized by the number k of robots which guarantees an additive error of $\mathcal{O}(k^5)$.*

4 An FPT Algorithm for GCMP1 Parameterized by k

The aim of this section is to establish Theorem 1 by using Theorem 15.

► **Theorem 1.** *GCMP1 is FPT parameterized by the number k of robots.*

Proof. Let $\mathcal{I} = (G, \mathcal{R} = (\mathcal{M}, \mathcal{F}), k, \ell)$ be an instance of GCMP1. By Statement 1 of Theorem 15, in FPT-time we can compute an equivalent instance $\mathcal{I}_j = (G_j, \mathcal{R}_j = (\mathcal{M}_j, \emptyset), k_j, \ell_j)$ to $\mathcal{I}$ satisfying that $k_j \leq k$ and $|\text{opt}(\mathcal{I}_j)| = \text{dist}_{G_j}(s_1, t_1) + \mathcal{O}(k^5)$, where $\mathcal{M} = \{R_1\}$. Moreover, for every robot $R_i \in \mathcal{M}_j \setminus \{R_1\}$, the moves of R_i in $\text{opt}(\mathcal{I}_j)$ are restricted to the vertices of a vertex-set $B(R_i)$ of cardinality at most $k^{\mathcal{O}(k^4)}$ that is computable in linear time.

Define a state graph $\mathcal{Q}$ whose vertices are k -tuples with coordinates defined as follows. The first coordinate of a k -tuple corresponds to R_1 and can be any of the at most n vertices in $V(G_j)$; the i -th coordinate of a tuple, where $i \in \{2, \dots, k\}$, encodes the possible location of R_i and is confined to the vertices in $B(R_i)$. We purge any tuple in which a vertex in $V(G_j)$ appears in more than one coordinate of the tuple (i.e., is occupied by more than one robot in the same time step). Since $|B(R_i)| = k^{\mathcal{O}(k^4)}$, for $i \in \{2, \dots, k\}$, it follows that the number of vertices in $\mathcal{Q}$ is at most $\mathcal{O}(n \cdot k^{\mathcal{O}(k^5)})$. Two vertices/states S and S' in $\mathcal{Q}$ are adjacent if there is a valid (i.e., causing no collision) single (parallel) move for the robots from their locations in S to their locations in S' ; the weight of an edge (S, S') in $\mathcal{Q}$ is the number of robots in S that have moved (i.e., their positions have changed).

Define the starting configuration S_{start} in $\mathcal{Q}$ to be the k -tuple whose coordinates correspond to the starting positions of the robots in $\mathcal{I}_j$. Define S_{final} to be the k -tuple whose coordinates correspond to the destinations of the robots in $\mathcal{I}_j$. Now compute a shortest (weighted) path from S_{start} to S_{final} in $\mathcal{Q}$ (e.g., using Dijkstra's algorithm) and accept the instance $\mathcal{I}$ if and only if the weight of the computed shortest path is at most ℓ_j . The running

time of the algorithm is $\mathcal{O}(|V(\mathcal{Q})|^2) = \mathcal{O}(n^2 \cdot k^{\mathcal{O}(k^5)})$, and it is clear that the above algorithm decides the instance correctly. It follows that GCMP1 is FPT parameterized by k . ◀

5 An FPT Algorithm Parameterized by Treewidth and k

In this section, we present an FPT algorithm for GCMP parameterized by the number of robots and the treewidth of the input graph combined. This result implies that the problem is FPT on certain graph classes such as graphs of bounded outerplanarity, which include subgrids of bounded height. In this sense, the result can be seen as complementary to the recently established NP-hardness of the problem on bounded-height subgrids [20].

5.1 Terminology and Observations

We begin by setting up the required terminology and proving some observations used for our algorithm.

► **Definition 16 (Semi-routes).** A *semi-route* for R_i is a tuple $W_i = (u_0^i, \dots, u_t^i)$ such that each u_j^i is either a vertex of G or the symbol $\perp$, and such that (i) $u_0^i = s_i$ and moreover, if $R_i \in \mathcal{M}$, then $u_t^i = t_i$, and (ii) $\forall j \in [t]$, either $u_{j-1}^i = u_j^i$ or $u_{j-1}^i u_j^i \in E(G)$ or one out of u_{j-1}^i and u_j^i is the symbol $\perp$. The notion of two conflicting semi-routes is identical to that for routes, except that we only consider time steps where neither semi-routes has $\perp$. Formally, two semi-routes $W_i = (u_0^i, \dots, u_t^i)$ and $W_j = (u_0^j, \dots, u_t^j)$, where $i \neq j \in [k]$, are *non-conflicting* if (i) $\forall r \in \{0, \dots, t\}$, either $u_r^i = \perp$ or $u_r^j = \perp$ or $u_r^i \neq u_r^j$, and (ii) $\nexists r \in \{0, \dots, t-1\}$ such that $u_{r+1}^j = u_r^i$ and $u_{r+1}^i = u_r^j$ and $\perp \notin \{u_r^i, u_{r+1}^i, u_r^j, u_{r+1}^j\}$. Otherwise, we say that W_i and W_j *conflict*.

Intuitively, the above definition gives us a notion of partial routes, where the robots can be thought of as having become “invisible” (but still potentially in motion) for some time steps during a route (assume only a small part of the graph is visible to us). These time steps where a robot disappears from view are represented by the symbol $\perp$. Note that a robot can “reappear” at a different vertex than the one it “disappeared” at. A *semi-schedule* for $\mathcal{R}$ is a set of semi-routes during a time interval $[0, t]$ that are pairwise non-conflicting. The *length* of a semi-route and a semi-schedule is naturally defined as the number of time steps in which the robot moves from one vertex to a different vertex, and the *total traveled length* of a semi-schedule is the sum of the lengths of its semi-routes.

In what follows, let $\mathcal{I} = (G, \mathcal{R} = (\mathcal{M}, \mathcal{F}), k, \ell)$ be an instance of GCMP. Let (H, Z) be a p -boundaried subgraph of G with boundary Z containing all the terminals. Let S be a schedule for this instance with routes $W_i, i \in [k]$.

► **Definition 17 (Signatures of Schedules).** Call the tuples in the set $(Z \cup \{\uparrow, \downarrow\})^k$, *configuration tuples* for (H, Z) . We define the *signature* of the schedule S with respect to (H, Z) as a sequence of tuples $\tau_0, \dots, \tau_t$, where each τ_i is a configuration tuple defined as follows: the j^{th} coordinate of τ_i signifies whether at the end of time step i

- R_j is on a vertex $v \in Z$, in which case this value is v ; or
- whether it is inside H but not on the boundary, in which case this value is $\downarrow$; or
- whether it is disjoint from H , in which case this value is $\uparrow$.

The tuple τ_i is called the *signature of the schedule S with respect to (H, Z) at time step i* . The tuple τ_0 and τ_t are called *starting* and *ending* configuration tuples.

► **Definition 18 (Checkpoints of Schedules).** We say that the schedule S :

- *externally affects* (H, Z) at time steps i and $i + 1$ if some robot moves from a vertex outside H to a vertex in Z during time step $i + 1$, or if some robot moves from some vertex of Z to a vertex outside H during time step $i + 1$. That is, for some robot R_j , we either have that $u_i^j \notin V(H)$ and $u_{i+1}^j \in Z$, or we have $u_i^j \in Z$ and $u_{i+1}^j \notin V(H)$.
- *internally affects* (H, Z) at time steps i and $i + 1$ if some robot moves from a vertex of H to a vertex of Z during time step $i + 1$, or if some robot moves from some vertex of Z to a vertex in H during time step $i + 1$. That is, for some robot R_j , we have $u_i^j \neq u_{i+1}^j$ and moreover, we either have that $u_i^j \in V(H)$ and $u_{i+1}^j \in Z$ or we have $u_i^j \in Z$ and $u_{i+1}^j \in V(H)$.
- *affects* (H, Z) at time steps i and $i + 1$ if it internally or externally affects (H, Z) at time steps i and $i + 1$.

The pairs of consecutive time steps at which the schedule affects (H, Z) are called *checkpoints of the schedule with respect to* (H, Z) . We drop the explicit reference to (H, Z) whenever it is clear from the context. Two checkpoints $(x, x + 1)$ and $(y, y + 1)$ are said to be *consecutive* if $x < y$ and there is no checkpoint $(z, z + 1)$ such that $x < z < y$.

Roughly speaking, the checkpoints are the time steps at which the schedule interacts in some way with the separator Z . The interaction involves a robot either moving to or from a vertex of Z . As a result, the notion of checkpoints enables one to “decompose” a solution along the vertices of Z between consecutive checkpoints. We will argue that whenever Z is sufficiently small, the number of possible checkpoints will also be small.

► **Definition K** (*v*-Rigid Pairs). Let $v \in Z$. A pair of configuration tuples (τ_1, τ_2) is a *v-rigid pair* if there is a unique coordinate j in which these tuples differ, exactly one out of $\tau_1[j], \tau_2[j]$ is equal to v and exactly one out of $\tau_1[j], \tau_2[j]$ is equal to $\perp$.

The above definition is useful when considering checkpoints where the schedule internally affects (H, Z) with the restriction that exactly one robot moves into Z from $V(H) \setminus Z$ or from Z into $V(H) \setminus Z$.

► **Definition 19** (Semi-schedules). A *semi-schedule with respect to* (H, Z) is a semi-schedule where every vertex on every semi-route is contained in $V(H)$ and for every subtuple $(u, \perp, \dots, \perp, v)$ in a semi-route, where $u, v \in V(H)$, it must be the case that $u, v \in Z$.

The intuition here is that whenever the robot “vanishes” or “reappears”, it happens at the boundary Z . This allows us to analyze how a schedule interacts with the subgraph H since every movement between H and the rest of G must happen through the boundary Z .

► **Definition 20** (Signatures of Semi-schedules). The *signature* $\tau_0, \dots, \tau_t$ of a semi-schedule with respect to (H, Z) is defined similarly to that of a schedule after making the assumption that $\perp$ denotes a vertex outside H , that is, we have the symbol $\uparrow$ in the j^{th} coordinate of tuple τ_i if and only if the robot R_j has $u_i^j = \perp$. A semi-schedule *externally affects* (H, Z) at time steps i and $i + 1$ if some robot “moves” from $\perp$ to Z during time step $i + 1$ or it “moves” from Z to $\perp$ during time step $i + 1$. That is, for some robot R_j , we have $u_i^j = \perp$ and $u_{i+1}^j \in Z$ or we have $u_i^j \in Z$ and $u_{i+1}^j = \perp$. The notions of internally affecting (H, Z) , affecting (H, Z) , checkpoints with respect to (H, Z) are defined exactly as for schedules.

► **Definition 21** (Checkpoint Tuples). *Checkpoint tuples* of a schedule or semi-schedule S with respect to (H, Z) are defined as those configuration tuples for (H, Z) that appear in the signature of S at the time steps participating in checkpoints of S with respect to (H, Z) . The *checkpoint tuple sequence* of S with respect to (H, Z) is the set of checkpoint tuples ordered according to their respective time steps.

Our dynamic programming algorithm will make use of the following observation about signatures of schedules and semi-schedules to narrow down the search space.

► **Observation 22.** *Let $\tau_0, \dots, \tau_t$ be the signature of a schedule or semi-schedule with respect to (H, Z) . Then, the following properties hold:*

1. $\tau_0 = (s_1, \dots, s_k)$. Moreover, the coordinates of τ_t that correspond to the robots in $\mathcal{M}$ are fixed, that is, $\tau_t = (t_1, \dots, t_{|\mathcal{M}|}, v_1, \dots, v_{k-|\mathcal{M}|})$ for some vertices $v_1, \dots, v_{k-|\mathcal{M}|}$ distinct from $\{t_1, \dots, t_{|\mathcal{M}|}\}$.
2. The first two and last two tuples in the signature are at checkpoints. That is, $(0, 1)$ and $(t-1, t)$ are checkpoints.
3. Let $(x, x+1), (y, y+1)$ be consecutive checkpoints such that $x+1 < y$. Then, the tuples $\tau_{x+2}, \dots, \tau_{y-1}$ are identical.
4. Let $(x, x+1)$ be a checkpoint. There is a coordinate j such that $\tau_x[j] \neq \tau_{x+1}[j]$ and at least one of $\tau_x[j]$ or $\tau_{x+1}[j]$ is a vertex of Z .
5. Let $(x, x+1)$ be a checkpoint. There is no coordinate j such that $\tau_x[j] = \uparrow$ and $\tau_{x+1}[j] = \downarrow$ or $\tau_x[j] = \downarrow$ and $\tau_{x+1}[j] = \uparrow$.
6. For every time step x , τ_x contains at most one occurrence of any vertex of Z .
7. Let $(x, x+1)$ be a checkpoint and suppose that $\tau_x[j] \neq v \in Z$ and $\tau_{x+1}[j] = v$. Then, either there is no j' such that $\tau_x[j'] = v$ or it must be the case that $\tau_{x+1}[j'] \neq v$.
8. Parallel movements between vertices of the boundary Z in a single time step must happen along edges that exist and moreover, no edge can be used by two robots.

Proof. The first two statements follow from the fact that the terminals are contained in Z . The third, fourth and fifth statements follow from the definition of checkpoints and the fact that the boundary is a separator between $V(H) \setminus Z$ and $V(G) \setminus V(H)$. The sixth and seventh statements are due to the fact that no vertex can be occupied by two robots at the same time. The eighth statement is due to the fact that a schedule or semi-schedule must be comprised of pairwise non-conflicting routes or semi-routes. ◀

5.2 Using Bounded Checkpoints to Solve the Problem on Bounded Treewidth Graphs

► **Observation 23.** *Let $\mathcal{I} = (G, \mathcal{R} = (\mathcal{M}, \mathcal{F}), k, \ell)$ be an instance of GCMP. Let (H, Z) be a p -boundaried subgraph of G with boundary Z containing all the terminals. If there is a schedule S for $\mathcal{I}$ in which each vertex is entered by any robot at most $\nu(k)$ times, then the following hold.*

1. The number of checkpoints of S with respect to (H, Z) is bounded by $\mathcal{O}(pk \cdot \nu(k))$.
2. The number of possible checkpoint tuple sequences of S with respect to (H, Z) is bounded by $g(k, p) = p^{\mathcal{O}(pk \cdot \nu(k))}$.

Proof. From Definition 18, we have that the number of checkpoints of S with respect to (H, Z) is bounded by $\mathcal{O}(pk \cdot \nu(k))$. This is because each of the p vertices in Z can be visited at most $\nu(k)$ times by any of the k robots. Moreover, the number of possible choices for a specific checkpoint tuple is bounded by $(p+2)^k$. This is simply because these are configuration tuples. This gives us the stated bound on the number of possible checkpoint tuple sequences of S with respect to (H, Z) . ◀

We now proceed by defining the partial solutions which play a central role in our algorithm.

► **Definition 24** (Partial Solutions). Let $\mathcal{I} = (G, \mathcal{R} = (\mathcal{M}, \mathcal{F}), k, \ell)$ be an instance of GCMP. Let (H, Z) be a p -boundaried subgraph of G with boundary Z containing all the terminals. Given an ordered set of configuration tuples $\gamma_1, \dots, \gamma_q$ for (H, Z) where q is even, γ_1 is a starting configuration tuple and γ_q is an ending configuration tuple, a *partial solution corresponding to the sequence* $(\gamma_1, \gamma_2), \dots, (\gamma_{q-1}, \gamma_q)$ is a semi-schedule S with respect to (H, Z) such that the checkpoint tuple sequence of S with respect to (H, Z) is exactly these tuples in this order. That is, the checkpoints of S with respect to (H, Z) are $(x_1, x_2), \dots, (x_{q-1}, x_q)$ and $\tau_{x_i} = \gamma_i$ for each $i \in [q]$.

5.2.1 Dynamic Programming over Tree Decompositions

► **Theorem 2.** GCMP is FPT parameterized by the number of robots and the treewidth of the input graph.

Proof. Let $\mathcal{I} = (G, \mathcal{R} = (\mathcal{M}, \mathcal{F}), k, \ell)$ be the given instance of GCMP. We assume that a schedule exists. This can be checked in polynomial time [42]. Let ρ denote the upper bound on the size of an optimal schedule given by our additive approximation. If $\ell \geq \rho$, then we are done. So assume this is not the case.

We start with a nice tree decomposition (T, β) of G of optimal width. This can be computed using Proposition A. Now, add the terminals to every bag. Call the resulting tree decomposition (T, β') and let its width be w . Note that although we have added the terminals to all the bags, the tree T has remained the same. Hence, we just have a slightly modified version of nice tree decompositions, where the root bag and every leaf bag is exactly the set of terminals. However, the remaining nodes (introduce, forget, join) are as described in Definition 6. Moreover, we have the mapping $\gamma' : V(T) \rightarrow 2^{V(G)}$ defined as $\gamma'(t) = \bigcup_{u \preceq t} \beta'(u)$.

By invoking Theorem 3 and setting $\nu(k) = \mathcal{O}(k^5)$ in Observation 23, we infer that the length of the checkpoint tuple sequence of S with respect to each boundaried graph $(G_{x,T}^\downarrow, \beta'(x))$ where $x \in V(T)$ is at most twice the number of checkpoints in this sequence and so, is bounded by $\lambda(k, w) = \mathcal{O}(wk^6)$. Moreover, by the same observation, the number of possible checkpoint tuple sequences of S with respect to each boundaried graph $(G_{x,T}^\downarrow, \beta'(x))$ where $x \in V(T)$, is at most $g(k, w + 1) = w^{\mathcal{O}(wk^6)}$.

Based on this fact, our goal is to compute, for every $x \in V(T)$ and boundaried graph $(G_{x,T}^\downarrow, \beta'(x))$, and for every possible checkpoint tuple sequence of length at most $\lambda(k, w)$ with respect to this boundaried graph, the length of the best partial solution corresponding to this checkpoint tuple sequence within the boundaried graph. For “infeasible” checkpoint tuple sequences, we assign them a length of $\rho + 1$ to indicate infeasibility. When x is the root node, $G_{x,T}^\downarrow$ is exactly the input graph G . Hence, the solution is given by the minimum entry in the table computed at the root node. Note that we only describe an algorithm to compute the length of an optimal solution. However, it will be straightforward to see that an optimal schedule can also be produced in the same running time.

Consider a node $x \in V(T)$ and let $\mathcal{C}_x$ denote the set of possible configuration tuples over $\beta'(x)$. That is, $\mathcal{C}_x = (\beta'(x) \cup \{\uparrow, \downarrow\})^k$. For every sequence $\sigma = ((\tau_1, \tau_2), \dots, (\tau_s, \tau_{s+1}))$ where each $\tau_i \in \mathcal{C}_x$, we say that σ is a *sequence at x* . Moreover, σ is called *good* if Properties 1-2 and Properties 4-8 listed in Observation 22 hold. Any other sequence at x is *bad*. The length of the sequence σ described above is $s + 1$.

For every good sequence σ at x of length at most $\lambda(k, w)$, we define h_σ^x to be the value of the best partial solution corresponding to σ in the boundaried graph $(G_{x,T}^\downarrow, \beta'(x))$. For every bad sequence σ at x of length at most $\lambda(k, w)$, h_σ^x is defined as $\rho + 1$. This is just to indicate

that we do not look for partial solutions corresponding to these sequences as satisfying the properties in Observation 22 is a necessary condition for a sequence to correspond to checkpoint tuples. Note that for a given sequence σ of length at most $\lambda(k, w)$, it is straightforward to check the relevant properties from Observation 22 and thus determine whether it is good or bad in time $\lambda(k, w)n^{\mathcal{O}(1)}$.

We next proceed to the description of the computation of our dynamic programming table. Every bad sequence σ of length at most $\lambda(k, w)$ is identified at every bag and h_σ is set to $\rho + 1$. As standard, we next proceed bottom up, starting from the leaf nodes.

Leaf node. At each leaf node x , and every good sequence σ at x of length at most $\lambda(k, w)$, h_σ^x is computed by brute force since the graphs have at most $w + 1$ vertices.

Forget node. Let x be a forget node with child node y . Let v be the unique vertex in $\beta'(y) \setminus \beta'(x)$. Let σ be a good sequence of length at most $\lambda(k, w)$ at the node x . We compute h_σ^x as follows.

1. Let Σ_v denote the set of all sequences obtained from σ by going over all possible ways of substituting some occurrences of $\downarrow$ in σ with v .

Since there are at most $(w + 1) \cdot \lambda(k, w)$ occurrences of $\downarrow$ in σ , the size of the set Σ_v is bounded by $\lambda'(k, w) = 2^{(w+1) \cdot \lambda(k, w)}$.

The idea behind this step is to “guess”, in the partial solution corresponding to σ , all instances where a movement is made from a vertex in $\beta'(x)$ to the forgotten vertex v or from v to a vertex in $\beta'(x)$. Once this is done, we would still need to “guess” all instances where a movement is made from v to a vertex not in $\beta'(x)$ or vice-versa. Since x is a forget node, all such movements from v must go “below” to the vertices in $\gamma'(y) \setminus \beta'(y)$ and vice-versa. This guess is done next.

2. Let Σ'_v denote the set of all sequences obtained by going over every sequence σ' in Σ_v and inserting at most $\lambda(k, w)$ v -rigid pairs of tuples belonging to $\mathcal{C}_y$ (see Definition K) between the elements of σ' in all possible ways.

For each sequence σ' in Σ_v , the number of resulting sequences is at most $(k \cdot (w + 2)^k)^{\lambda(k, w)}$. This is because there are $2k \cdot (w + 2)^k$ possible v -rigid pairs (fix the coordinate of v and allow all possible entries in the remaining coordinates) and at most $\lambda(k, w)$ possible positions. Hence, the set Σ'_v has size at most $\lambda''(k, w) = \lambda'(k, w) \cdot (k \cdot (w + 2)^k)^{\lambda(k, w)}$.

As discussed in the last step, the goal of this step is to guess all steps where a movement is made from v to a vertex not in $\beta'(x)$ or in the other direction, in which case the robot must move to or from $\gamma'(y) \setminus \beta'(y)$. This is why we have the requirement of replacing v with $\downarrow$ or vice-versa in consecutive tuples, motivating Definition K. Moreover, we only need to guess pairs of tuples where *only* the coordinate that has v is changed, because all other pairs would already be in the sequence σ and all occurrences of v in these pairs guessed correctly in the previous step.

3. Inductively, for every good sequence σ' in Σ'_v of length at most $\lambda(k, w)$, we would have computed $h_{\sigma'}^y$ at the node y .

Hence, h_σ^x is defined to be the minimum $h_{\sigma'}^y$, where σ' ranges over all good sequences of length at most $\lambda(k, w)$ in the set Σ'_v .

The correctness is a consequence of the fact that our guesses in the first two steps exhaustively explore, for a partial solution S corresponding to σ , all possible checkpoint tuples of S with respect to $(G_{y,T}^\downarrow, \beta'(y))$.

Introduce node. Let x be an introduce node with child node y . Let v be the unique vertex in $\beta'(x) \setminus \beta'(y)$. Let σ be a good sequence of length at most $\lambda(k, w)$ at the node x . We compute h_σ^x as follows.

1. Let μ denote the total number of moves made in σ from v to vertices of $\beta'(y)$ or vice-versa. That is, μ denotes the number of pairs (τ_i, τ_{i+1}) in σ such that for some j , either $\tau_i[j] = v$ and $\tau_{i+1}[j] = v'$ for some $v' \in \beta'(y)$ or $\tau_i[j] = v'$ and $\tau_{i+1}[j] = v$ for some $v' \in \beta'(y)$.
The idea behind this step is to separate the required partial solution corresponding to σ into two parts – one part for all the moves between v and $\beta'(y)$ and the second part will then be a semi-schedule that is a partial solution corresponding to an appropriate sequence at the child node y . Notice that there cannot be a move made directly from v to vertices of $\gamma'(y) \setminus \beta'(y)$ because x is an introduce node and so, v is separated from $\gamma'(y) \setminus \beta'(y)$ by $\beta'(y)$.
2. Let σ' denote the sequence obtained from σ by replacing every occurrence of v with $\uparrow$ and remove all pairs of tuples that are identical. The idea in this step is to focus on the part of the partial solution corresponding to σ (call it S) that lies in the boundaried graph $(G_{y,T}^\downarrow, \beta'(y))$, which is a semi-schedule (call it S') with respect to $(G_{y,T}^\downarrow, \beta'(y))$. Since v is a vertex outside $(G_{y,T}^\downarrow)$, all occurrences of v must be treated as an “external” vertex and so, we replace these occurrences with $\uparrow$. If this results in a pair (τ_i, τ_{i+1}) where the tuples are identical, then this pair cannot be a pair of checkpoint tuples of S' with respect to $(G_{y,T}^\downarrow, \beta'(y))$ and so, we remove these.
3. Finally, h_σ^x is defined to be μ plus the value of $h_{\sigma'}^y$ at the node y .

Join node. Let x be a join node with children nodes y_1 and y_2 . Let σ be a good sequence of length at most $\lambda(k, w)$ at the node x . Let μ denote the total number of moves made in σ between vertices of $\beta'(y)$. That is, μ denotes the number of tuples $(u, v, \tau_i, \tau_{i+1}, j)$ where $u, v \in \beta'(y)$, (τ_i, τ_{i+1}) is a pair in σ , $\tau_i[j] = u$ and $\tau_{i+1}[j] = v$.

We now compute h_σ^x as follows.

1. Let Σ' denote the set of all sequences obtained from σ by going over every occurrence of $\downarrow$ in σ , and replacing it with one out of $\swarrow$ or $\searrow$.
The set Σ' has size at most $\lambda'(k, w)$ by the same reasoning as that used in the first step of processing forget nodes.
The idea in this step is to “guess”, for every occurrence of a vertex of $\gamma'(x) \setminus \beta'(x)$ in σ (which is indicated by $\downarrow$), whether this vertex is in $\gamma'(y_1) \setminus \beta'(y_1)$ (denoted by $\swarrow$) or in $\gamma'(y_2) \setminus \beta'(y_2)$ (denoted by $\searrow$) in some partial solution S corresponding to σ .
2. Let Σ'' denote pairs (σ_1, σ_2) of sequences obtained as follows. For every $\sigma' \in \Sigma'$ do:
 - 2.1. In σ' , replace all occurrences of $\swarrow$ with $\downarrow$ and replace all occurrences of $\searrow$ with $\uparrow$. Call the result σ_1 .
 - 2.2. In σ' , replace all occurrences of $\swarrow$ with $\uparrow$ and replace all occurrences of $\searrow$ with $\downarrow$. Call the result σ_2 .
 - 2.3. Add the pair (σ_1, σ_2) to Σ'' .

The idea in this step is to generate, for every guess made in the first step, two sequences σ_1 and σ_2 , expressing the checkpoint tuple sequences of S with respect to the boundaried graphs $(G_{y_1,T}^\downarrow, \beta'(y_1))$ and $(G_{y_2,T}^\downarrow, \beta'(y_2))$, respectively. Hence, to obtain σ_1 , the vertices “guessed” to be below $\beta'(y_2)$ are denoted by $\uparrow$ and the vertices “guessed” to be below $\beta'(y_1)$ are denoted by $\downarrow$. Similarly, to obtain σ_2 , the vertices “guessed” to be below $\beta'(y_1)$ are denoted by $\uparrow$ and the vertices “guessed” to be below $\beta'(y_2)$ are denoted by $\downarrow$.

We next use the fact that the length of S can be expressed as the sum of the lengths of partial solutions S_1 and S_2 corresponding to σ_1 and σ_2 minus the number of movements made between vertices of $\beta'(x)$ to avoid double counting these moves.

3. Hence, h_σ^x is defined to be the minimum over all $h_{\sigma_1}^{y_1} + h_{\sigma_2}^{y_2} - \mu$ where we range over all pairs $(\sigma_1, \sigma_2) \in \Sigma''$ where both σ_1 and σ_2 are good sequences. If such σ_1 and σ_2 do not exist, then h_σ^x is set to $\rho + 1$ to indicate infeasibility.

The number of entries at each bag of the tree decomposition as well as the time required to compute the table at each bag is clearly FPT in k and w . This completes the proof of Theorem 2. $\blacktriangleleft$

6 Using the Total Energy as the Parameter

In the final part of our paper, we consider the complexity of GCMP in settings where we are dealing with many robots but the energy budget ℓ is small. While intuitively this may seem like an “easier” case due to the fact that ℓ immediately bounds the number of robots that will end up moving in a hypothetical schedule, we show that the problem (and even its restriction to the often-studied case where all robots have destinations) is unlikely to be fixed-parameter tractable when parameterized by ℓ .

► **Theorem 4.** *GCMP is W[1]-hard when parameterized by ℓ .*

Proof. We provide a parameterized reduction from the well-known W[1]-complete MULTICOLORED CLIQUE problem [8, 12]: decide whether a given κ -partite graph $(V_1, \dots, V_\kappa, E)$ contains a clique of size κ . To avoid any confusion, we explicitly remark that in this proof we use κ to denote the parameter of the initial instance of MULTICOLORED CLIQUE and not the number of robots in the resulting instance of GCMP.

Our reduction takes an instance $(V_1, \dots, V_\kappa, E)$ of MULTICOLORED CLIQUE and constructs an instance $(G, \mathcal{R} = (\mathcal{M}, \mathcal{F}), \kappa', \ell)$ of GCMP as follows. To obtain G , we:

1. subdivide each edge $e \in E$ κ^3 many times;
2. attach a single new pendant vertex v' to each vertex $v \in V_1 \cup \dots \cup V_\kappa$ in the original graph; and
3. for each pair of integers i, j such that $1 \leq i < j \leq \kappa$, construct a new vertex $s_{i,j}$ and make it adjacent to every vertex in V_i , and construct a new vertex $t_{i,j}$ and make it adjacent to every vertex in V_j .

For $\mathcal{R} = (\mathcal{M}, \mathcal{F})$, we set $\mathcal{F} = \emptyset$ and add two sets of robots into $\mathcal{M}$. First, for each vertex v in $V_1 \cup \dots \cup V_\kappa$, we construct a *blocking* robot r^v and set v as its starting point as well as its destination. Second, for each of the newly constructed $s_{i,j} \in V(G)$, we construct a *clique* robot $r^{i,j}$, set $s_{i,j}$ as its starting point and $t_{i,j}$ as its destination. Finally, we set $\ell := 2\kappa + \binom{\kappa}{2} \cdot (\kappa^3 + 3)$ and $\kappa' := |\mathcal{R}|$. This completes the description of our reduction.

Towards proving correctness, we first observe that a hypothetical schedule for $(G, \mathcal{R}, \kappa', \ell)$ can never use less than $\ell := 2\kappa + \binom{\kappa}{2} \cdot (\kappa^3 + 3)$ travel length. Indeed, there are $\binom{\kappa}{2}$ many clique robots, and the shortest path between their starting and destination points has length $\kappa^3 + 3$. Moreover, every schedule must route each clique robot $r^{i,j}$ through at least one vertex in V_i and at least one vertex in V_j , and hence a hypothetical solution must spend a length of at least 2 to move at least one of the blocking robots in each V_p , $p \in [\kappa]$, out of the way and then back to its initial position (which is also its destination).

Now, assume that $(G, \mathcal{R}, \kappa', \ell)$ is a yes-instance. By the argument above, this implies that there is a schedule which moves precisely one of the blocking robots in each V_p , $p \in [\kappa]$; let us denote by w_p the starting vertex of the unique blocking robot which is moved from

V_p . Since $(G, \mathcal{R}, \kappa', \ell)$ is a yes-instance, E must contain an edge between each w_i and w_j , $1 \leq i < j \leq \kappa$; in particular, these vertices induce a clique in $(V_1, \dots, V_\kappa, E)$.

For the converse, assume that $(V_1, \dots, V_\kappa, E)$ contains a κ -clique $w_1, \dots, w_\kappa$. We construct a solution for $(G, \mathcal{R}, \kappa', \ell)$ as follows. First, for each robot starting at w_i , $i \in [\kappa]$, we spend a length of 1 to move it to the pendant vertex w'_i . Next, we route each robot $r^{i,j}$ from $s_{i,j}$ to w_i , through the $(\kappa^3 + 1)$ -length path to w_j (whereas the existence of this path is guaranteed by the existence of the edge $w_i w_j \in E$), and then to its destination $t_{i,j}$. Finally, we route each blocking robot which originally started at w_i back to w_i from w'_i . The travelled length of this schedule is precisely $2\kappa + \binom{\kappa}{2} \cdot (\kappa^3 + 3)$, as desired. ◀

As mentioned in the introduction, we complement Theorem 4 with a fixed-parameter algorithm on graph classes of bounded local treewidth.

Before proceeding to the proof, we recall the syntax of Monadic Second Order Logic (MSO) of graphs. It contains the logical connectives $\vee, \wedge, \neg, \Leftrightarrow, \Rightarrow$, variables for vertices, edges, sets of vertices and sets of edges, and the quantifiers $\forall$ and $\exists$, which can be applied to these variables. It also contains the following binary relations: (i) $u \in U$, where u is a vertex variable and U is a vertex set variable; (ii) $d \in D$, where d is an edge variable and D is an edge set variable; (iii) $\text{inc}(d, u)$, where d is an edge variable, u is a vertex variable, with the interpretation that the edge d is incident to u ; (iv) equality of variables representing vertices, edges, vertex sets and edge sets.

The well-known Courcelle's theorem [2, 7] states that checking whether a given graph G models a given MSO formula ϕ can be done in FPT time parameterized by the treewidth of G and the size of ϕ . Moreover, this result holds even if some vertices of G are given labels or colors (i.e., we allow a fixed number of additional unary relations over $V(G)$). This is because one can produce an equivalent graph G' such that G has bounded treewidth if and only if G' does, plus an alternate MSO formula ϕ' such that G models ϕ if and only if G' models ϕ' .

► **Theorem 5.** *GCMP is FPT parameterized by ℓ on graph classes of bounded local treewidth.*

Proof. Let $\mathcal{I} = (G, \mathcal{R} = (\mathcal{M}, \mathcal{F}), k, \ell)$ be the given instance of GCMP where G belongs to a graph class that is closed under taking subgraphs and in which the treewidth of a graph of diameter d is bounded by $f(d)$ for some function f . Let $\mathcal{M}'$ denote the set of those robots in $\mathcal{M}$ whose final position is distinct from the starting position. Since each of the robots in $\mathcal{M}'$ must move at least one step, it follows that if $|\mathcal{M}'| > \ell$, then the instance is a no-instance. So, assume that $|\mathcal{M}'| \leq \ell$. Let U denote the set of starting points of the robots in $\mathcal{M}'$.

Consider an optimal schedule S . We say that an edge e appears in a route $W = (u_0, \dots, u_t)$ in S if the endpoints of e are the vertices u_{j-1} and u_j for some $j \in [t]$. We say that e appears in S if it appears in some route in S . Let E_S denote the set of edges of G that appear in S . We observe that every connected component of the subgraph of G induced by E_S (call it G_S) contains a vertex of U . If this were not the case and there is a connected component of G_S disjoint from U , then we can delete all the routes whose edges appear in this connected component and still have a feasible schedule for the given instance, contradicting the optimality of S . Moreover, since S has total energy at most ℓ , it follows that at most ℓ edges can appear in S . Hence, it follows that S is contained in the subgraph H defined as the union of the subgraphs induced by the ℓ -balls centered at the starting positions of the robots in $\mathcal{M}'$.

Notice that since H is obtained by taking the union of at most ℓ graphs, each of diameter at most 2ℓ , it follows that H has diameter at most $2\ell^2$ and so, the treewidth of H is upper bounded by $f(2\ell^2)$. It remains to argue that GCMP is FPT parameterized by the energy

and treewidth of the input graph. We do this by making some guesses, obtaining a bounded number of MSO formulas such that input is a yes-instance if and only if at least one of these formulas is true on the graph H with a label function (with constant number of labels) and then invoking Courcelle's theorem [2, 7]. Let $\mathcal{M}''$ denote the subset of $\mathcal{M} \setminus \mathcal{M}'$ contained in H and let $\mathcal{R}' = \mathcal{R} \cap V(H)$.

Suppose that $\alpha \leq \ell$ robots move in S and let the routes in S be $\{W_i = (v_1^i, \dots, v_{d_i}^i) \mid i \in [\alpha]\}$. Assume that the first $|\mathcal{M}'|$ routes, $W_1, \dots, W_{|\mathcal{M}'|}$, correspond to the robots in $\mathcal{M}'$. We guess α and $d_1, \dots, d_\alpha$ as these are all at most ℓ . Moreover, we guess, for every pair of vertices v_j^i and v_q^p whether they are equal. Let this be expressed by a function $\mu(v_j^i, v_q^p)$ that evaluates to 1 if and only if these vertices are guessed to be equal. Thus, even though we do not know the actual vertices in the solution (except for the starting and endpoint positions of the robots in $\mathcal{M}'$), this guess gives us the “structure” of the entire solution. Notice that the number of guesses is bounded by a function of ℓ . Now, we consider the labeled graph H where the starting points of all robots in $\mathcal{R}' \setminus \mathcal{M}''$ are labeled red and the starting points of the robots in $\mathcal{M}''$ are labeled blue. Fix a guess of α , the numbers $d_1, \dots, d_\alpha$ and a guess of the function μ and express the following as an MSO formula.

There are ordered vertex sets $\{W_i = (v_1^i, \dots, v_{d_i}^i) \mid i \in [\alpha]\}$ such that the following hold:

1. Each W_i is a walk in H .
2. For every i, j, p, q , v_j^i and v_q^p are equal if and only if $\mu(v_j^i, v_q^p)$ is 1.
3. The walks W_i are pairwise non-conflicting routes.
4. For each of the first $|\mathcal{M}'|$ walks $W_1, \dots, W_{|\mathcal{M}'|}$, the initial and final vertices are precisely the starting and ending points, respectively, of some robot in $\mathcal{M}'$.
5. For each labeled vertex appearing on any walk W_i , this vertex must be the first vertex of some walk W_i .
6. For each walk W_i starting at a blue vertex, it must terminate at the same blue vertex.

Expressing the above in MSO is straightforward to do, so we do not go in to lower level details of the MSO formula.

We next argue that the input is a yes-instance if and only if there is a guess of α , the numbers $d_1, \dots, d_\alpha$ and a guess of the function μ such that the resulting MSO formula is true on H . Consider the forward direction. An optimal schedule S with routes $W_1, \dots, W_\alpha$ naturally gives us a “correct” guess of α , $d_1, \dots, d_\alpha$ and the function μ . Moreover notice that Properties 1-4 are clearly satisfied. For Property 5, notice that only the starting points of robots that never move can be disjoint from the set of initial vertices of the routes $\{W_i \mid i \in [\alpha]\}$ in the optimal schedule S . Since these robots do not move, they also cannot appear on any route W_i or they would obstruct a robot that moves. Finally, Property 6 is satisfied since every robot in $\mathcal{M}''$ that moves also ends up returning to its starting point. The converse is trivial. $\blacktriangleleft$

7 Conclusion

In this paper, we presented parameterized algorithms for fundamental coordinated motion planning problems on graphs. In particular, we proved that GCMP1 is fixed-parameter tractable parameterized by the number of robots, and that GCMP is fixed-parameter tractable parameterized by the number k of robots and the treewidth of the input graph combined. This latter result implies that GCMP is fixed-parameter tractable in several graph classes which may be of interest w.r.t. application domains, including graphs of bounded outerplanarity.

We conclude by highlighting three directions that may be pursued in future works:

1. Is GCMP FPT parameterized by k (alone) on planar graphs, or even on general graphs?
2. What is the complexity of GCMP on trees? While GCMP1 is known to be in P on trees, the complexity of the general problem on trees remains unresolved.
3. The focus of this paper was on minimizing the travel length/energy, which is one of the most-studied optimization objectives. Nevertheless, there are other important objectives, most notably that of minimizing the makespan. It seems that at least in the settings considered here, optimizing the makespan may be a computationally more challenging problem. In particular, the question of whether COORDINATED MOTION PLANNING on trees is FPT parameterized by k remains open.

References

- 1 Aviv Adler, Mark de Berg, Dan Halperin, and Kiril Solovey. Efficient multi-robot motion planning for unlabeled discs in simple polygons. *IEEE Transactions on Automation Science and Engineering*, 12(4):1309–1317, 2015.
- 2 Stefan Arnborg, Jens Lagergren, and Detlef Seese. Easy problems for tree-decomposable graphs. *Journal of Algorithms*, 12:308–340, 1991.
- 3 Jacopo Banfi, Nicola Basilico, and Francesco Amigoni. Intractability of time-optimal multirobot path planning on 2D grid graphs with holes. *IEEE Robotics and Automation Letters*, 2(4):1941–1947, 2017.
- 4 Bahareh Banyassady, Mark de Berg, Karl Bringmann, Kevin Buchin, Henning Fernau, Dan Halperin, Irina Kostitsyna, Yoshio Okamoto, and Stijn Slot. Unlabeled multi-robot motion planning with tighter separation bounds. In *SoCG*, volume 224, pages 12:1–12:16, 2022.
- 5 Eli Boyarski, Ariel Felner, Roni Stern, Guni Sharon, David Tolpin, Oded Betzalel, and Solomon Eyal Shimony. ICBS: Improved conflict-based search algorithm for multi-agent pathfinding. In *IJCAI*, pages 740–746, 2015.
- 6 Josh Brunner, Lily Chung, Erik D. Demaine, Dylan H. Hendrickson, Adam Hesterberg, Adam Suhl, and Avi Zeff. 1×1 rush hour with fixed blocks is PSPACE-complete. In *FUN*, volume 157, pages 7:1–7:14, 2021.
- 7 Bruno Courcelle. The monadic second-order logic of graphs. i. recognizable sets of finite graphs. *Inf. Comput.*, 85(1):12–75, 1990. doi:10.1016/0890-5401(90)90043-H.
- 8 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.
- 9 Erik D. Demaine, Sándor P. Fekete, Phillip Keldenich, Henk Meijer, and Christian Scheffer. Coordinated motion planning: Reconfiguring a swarm of labeled robots with bounded stretch. *SIAM Journal on Computing*, 48(6):1727–1762, 2019.
- 10 Erik D. Demaine and Mikhail Rudoy. A simple proof that the $(n^2 - 1)$ -puzzle is hard. *Theoretical Computer Science*, 732:80–84, 2018.
- 11 Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.
- 12 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013.
- 13 Adrian Dumitrescu. Motion planning and reconfiguration for systems of multiple objects. In Sascha Kolski, editor, *Mobile Robots*, chapter 24. IntechOpen, Rijeka, 2007.
- 14 Eduard Eiben, Robert Ganian, and Iyad Kanj. The parameterized complexity of coordinated motion planning. In *SoCG*, volume 258, pages 28:1–28:16, 2023.
- 15 David Eppstein. Diameter and treewidth in minor-closed graph families. *Algorithmica*, 27(3):275–291, 2000.
- 16 Sándor P. Fekete, Phillip Keldenich, Dominik Krupke, and Joseph S. B. Mitchell. Computing coordinated motion plans for robot swarms: The CG: SHOP challenge 2021. *ACM Journal on Experimental Algorithmics*, 27:3.1:1–3.1:12, 2022.

- 17 Foivos Fioravantes, Dusan Knop, Jan Matyás Kristan, Nikolaos Melissinos, and Michal Opler. Exact algorithms and lowerbounds for multiagent pathfinding: Power of treelike topology. *CoRR*, abs/2312.09646, 2023.
- 18 Gary William Flake and Eric B. Baum. Rush hour is PSPACE-complete, or “why you should generously tip parking lot attendants”. *Theoretical Computer Science*, 270(1-2):895–911, 2002.
- 19 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*, volume XIV of *Texts in Theoretical Computer Science. An EATCS Series*. Springer, Berlin, 2006.
- 20 Tzvika Geft and Dan Halperin. Refined hardness of distance-optimal multi-agent path finding. In *AAMAS*, page 481–488, 2022.
- 21 Oded Goldreich. *Finding the shortest move-sequence in the graph-generalized 15-puzzle is NP-hard*. Springer, 2011.
- 22 Martin Grohe. Local tree-width, excluded minors, and approximation algorithms. *Combinatorica*, 23(4):613–632, 2003.
- 23 Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.
- 24 John E. Hopcroft, Wolfgang J. Paul, and Leslie G. Valiant. On time versus space and related problems. In *FOCS*, pages 57–64. IEEE Computer Society, 1975.
- 25 John E. Hopcroft, Jacob T. Schwartz, and Micha Sharir. On the complexity of motion planning for multiple independent objects; PSPACE-hardness of the “Warehouseman’s Problem”. *The International Journal of Robotics Research*, 3(4):76–88, 1984.
- 26 Tuukka Korhonen. A single-exponential time 2-approximation algorithm for treewidth. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pages 184–192. IEEE, 2021. doi:10.1109/FOCS52979.2021.00026.
- 27 Paul Liu, Jack Spalding-Jamieson, Brandon Zhang, and Da Wei Zheng. Coordinated motion planning through randomized k -Opt (CG challenge). In *SoCG*, volume 189, pages 64:1–64:8, 2021.
- 28 Christos H. Papadimitriou, Prabhakar Raghavan, Madhu Sudan, and Hisao Tamaki. Motion planning on a graph (extended abstract). In *STOC*, pages 511–520, 1994.
- 29 Geetha Ramanathan and Vangalur S. Alagar. Algorithmic motion planning in robotics: Coordinated motion of several disks amidst polygonal obstacles. In *ICRA*, volume 2, pages 514–522, 1985.
- 30 Daniel Ratner and Manfred Warmuth. The $(n^2 - 1)$ -puzzle and related relocation problems. *Journal of Symbolic Computation*, 10(2):111–137, 1990.
- 31 John H Reif. Complexity of the mover’s problem and generalizations. In *FOCS*, pages 421–427, 1979.
- 32 Oren Salzman and Roni Stern. Research challenges and opportunities in multi-agent path finding and multi-agent pickup and delivery problems. In *AAMAS*, pages 1711–1715, 2020.
- 33 Jacob T. Schwartz and Micha Sharir. On the piano movers’ problem: III. coordinating the motion of several independent bodies: The special case of circular bodies moving amidst polygonal barriers. *The International Journal of Robotics Research*, 2:46 – 75, 1983.
- 34 Jacob T. Schwartz and Micha Sharir. On the “piano movers’” problem I. The case of a two-dimensional rigid polygonal body moving amidst polygonal barriers. *Communications on Pure and Applied Mathematics*, 36(3):345–398, 1983.
- 35 Jacob T. Schwartz and Micha Sharir. On the “piano movers” problem. II. General techniques for computing topological properties of real algebraic manifolds. *Advances in Applied Mathematics*, 4(3):298–351, 1983.
- 36 Guni Sharon, Roni Stern, Ariel Felner, and Nathan R. Sturtevant. Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence*, 219:40–66, 2015.
- 37 Irving Solis, James Motes, Read Sandström, and Nancy M. Amato. Representation-optimal multi-robot motion planning using conflict-based search. *IEEE Robotics Automation Letters*, 6(3):4608–4615, 2021.

- 38 Kiril Solovey and Dan Halperin. On the hardness of unlabeled multi-robot motion planning. *The International Journal of Robotics Research*, 35(14):1750–1759, 2016.
- 39 Glenn Wagner and Howie Choset. Subdimensional expansion for multirobot path planning. *Artificial Intelligence*, 219:1–24, 2015.
- 40 Jingjin Yu and Steven M. LaValle. Structure and intractability of optimal multi-robot path planning on graphs. In *AAAI*, pages 1443–1449, 2013.
- 41 Jingjin Yu and Steven M. LaValle. Optimal multirobot path planning on graphs: Complete algorithms and effective heuristics. *IEEE Transactions on Robotics*, 32(5):1163–1177, 2016.
- 42 Jingjin Yu and Daniela Rus. Pebble motion on graphs with rotations: Efficient feasibility tests and planning algorithms. In *WAFR*, volume 107 of *Springer Tracts in Advanced Robotics*, pages 729–746, 2014.