Dispersion, Capacitated Nodes, and the Power of a Trusted Shepherd

William K. Moses Jr.
wkmjr3@gmail.com
Durham University
Durham, UK

Amanda Redlich
amanda_redlich@uml.edu
University of Massachusetts Lowell
Lowell, USA

ABSTRACT

In this paper, we look at and expand the problems of dispersion and Byzantine dispersion of mobile robots on a graph, introduced by Augustine and Moses Jr. [ICDCN 2018] and by Molla, Mondal, and Moses Jr. [ALGOSENSORS 2020], respectively, to graphs where nodes have variable capacities. We use the idea of a single shepherd, a more powerful robot that will never act in a Byzantine manner, to achieve fast Byzantine dispersion, even when other robots may be strong Byzantine in nature. We also show the benefit of a shepherd for dispersion on capacitated graphs when no Byzantine robots are present.

CCS CONCEPTS

• **Theory of computation → Distributed algorithms**; • **Computing methodologies → Mobile agents**.

KEYWORDS

Mobile robots, Mobile agents, Capacitated graphs, Dispersion, Byzantine dispersion, Fault tolerance, Trusted shepherd

ACM Reference Format:
William K. Moses Jr. and Amanda Redlich. 2024. Dispersion, Capacitated Nodes, and the Power of a Trusted Shepherd. In *25th International Conference on Distributed Computing and Networking (ICDCN '24), January 4–7, 2024, Chennai, India.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3631461.3632310

1 INTRODUCTION

There are many instances in modern life where computational entities must move around in some space and work together with one another to perform some task. For example, self-driving cars interacting with one another in order to navigate intersections, overtake one another, and perform other driving behavior. Another example is that of using unmanned aerial vehicles to collect information about the weather [9]. These real world instances can be abstracted by using the model of mobile robots on a plane or on a graph.

In the context of mobile robots on a graph, several problems have been well studied in the past, such as exploration [5], gathering [4], scattering [8], and dispersion [3]. Of interest to us is the problem of dispersion of mobile robots on a graph, where k robots, initially arbitrarily placed on an n node graph, must move around such that each node has at most $\lceil k/n \rceil$ robots on it. This models, for example, k electric vehicles that each need a charge in a city with n charging stations. Since it may take far longer for a vehicle to charge up at a station than find a station, the goal is for as few vehicles as possible to share a station.

The problem of dispersion of mobile robots on a graph, introduced in [3], assumed that each node was identical with respect to satisfying the demand of the robot. However, in real life, this may not necessarily be the case. Consider again the electric vehicle example. Some locations may have multiple chargers while others have just one. Here we study the problem of dispersion on *capacitated* graphs to model this idea. Each node has a (possibly zero) capacity, and the total capacity across all nodes is at least the number of robots. Now dispersion is redefined, the robots must move around such that each node has at most *its capacity* of robots on it.

We further modify the problem to match real-world situations by introducing capacitated *Byzantine* dispersion. Robots can sometimes fail, either in simple ways (crash faults) or in arbitrary and unexpected ways (Byzantine faults). Byzantine dispersion was introduced and first studied in [15, 18]. In previous work [15, 16, 18], it was assumed that all robots were equally likely to be corrupted. However, in real life, the robots participating in a task may be heterogeneous [22]. Due to budget constraints or availability one may use a large number of inexpensive robots to perform a task and a single more expensive robot to aid these robots.

Thus, in this paper, we look at how a single powerful robot, which we call a *trusted shepherd*, and multiple weaker robots can solve Byzantine dispersion on a capacitated graph. We assume that this trusted shepherd will never act in a Byzantine manner and all robots are able to identify the shepherd. These are reasonable assumptions: The shepherd represents a special robot that may be more powerful than other robots. We can suppose that all robots have a sensor that is tuned to pick up a specific type of signal only emitted by the shepherd. For instance, if all robots are equipped with a light sensor but only the shepherd is equipped with a source of light, then it would be impossible for other robots to fake being the

Part of the work was done while William K. Moses Jr. was a post doctoral fellow at the University of Houston in Houston, USA. The work of William K. Moses Jr. was supported in part by NSF grants CCF-1540512, IIS-1633720, and CCF-1717075 and in part by BSF grant 2016419 and in part by UMass Lowell Pre-tenure Mathematics Faculty Seed Grant. The work of Amanda Redlich was supported in part by UMass Lowell Pre-tenure Mathematics Faculty Seed Grant.

ICDCN '24, January 4–7, 2024, Chennai, India
© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-1673-7/24/01…$15.00
https://doi.org/10.1145/3631461.3632310

shepherd.[1] As the shepherd represents a more powerful robot, with extra hardware or other mechanisms to prevent faulty behavior, it is reasonable to assume that the shepherd is not Byzantine.

1.1 Model

Graph Description. We consider a graph $G(V, E, c)$ with V the set of nodes, E the set of edges, and c a capacity function that maps each node $v \in V$ to some value in $[0, k^p]$, where k is the number of robots present in the graph and p is some positive constant.[2] We assume that $\sum_{v \in V} c(v) \geq k$. The nodes are anonymous, i.e., they do not have IDs. We use n to denote the number of nodes and m to denote the number of edges. Note that neither the graph structure nor c are known to robots prior to the start of the algorithm. A robot discovers $c(v)$ only when reaching v.

Robot Description. There are k (> 1) robots on the graph, each having a unique ID in the range $[1, k^q]$, where q is some positive constant. Among these robots, one of them is designated the *shepherd*. This robot can never be Byzantine. All robots can detect if a co-located robot is the shepherd.

We assume time proceeds in synchronous rounds. Each round consists of two steps: (i) robots co-located with each other communicate and each robot performs local computation, (ii) each robot stays at its current node or moves along an edge to an adjacent node. All robots start the prescribed algorithm at the same time.

Among the k robots, at most f of them may be strong Byzantine [7]. That is, they may deviate from prescribed algorithms, send incorrect information to other robots when communicating, and lie about their IDs. (This contrasts with weak Byzantine behavior [7], where robots may *not* lie about their IDs.) There is one subtlety with respect to strong Byzantine behavior that we wish to make explicit here. When the notion of a strong/weak Byzantine robot was introduced in [7] and was subsequently used in [16], it was assumed that when a robot is present at a node, it could see the labels of co-located robots, and all information exchanged is done in a "shouting" manner so that the information becomes common knowledge to all robots co-located on the node. This implicitly prevents a Sybil style attack, i.e., a strong Byzantine robot cannot pretend to send messages originating from multiple different robots. As this is implicitly taken care of by the model, we do not explicitly handle it in our algorithms.

When we consider the Byzantine setting in Section 2, as is usual (e.g. [7]) we assume that all robots have unlimited memory. When we consider the non-Byzantine setting in Section 3, we assume that all robots have limited memory with the exact values required for the given algorithm. Note that it is possible for robots to not know the value of a parameter (e.g., k) and yet require memory that is proportional to some function of that parameter (e.g., $\log k$) in order to perform some algorithm. This is a conditional guarantee of the algorithm and so long as the robots possess this minimum memory requirement, the algorithm will work as intended.

[1]Alternatively, one may assume that the shepherd can use cryptographic primitives to ensure that he is trusted, such as encrypting messages via a private key whose corresponding public key is known to all robots.
[2]Notice that some nodes may have zero capacity.

Global Knowledge and Assumptions. As a thumb rule, we assume that only the shepherd knows both the values of n and k unless otherwise stated. Regarding the algorithms for Byzantine dispersion, we assume that the robots do not know the value of f unless otherwise stated. While the explicit knowledge requirements are made clear in each Theorem, we also mention these requirements together in one place in Section 1.3.

Problem Statement: Dispersion on a Capacitated Graph: Given k robots initially placed arbitrarily on a capacitated graph of n nodes, the robots must re-position themselves autonomously to reach a configuration where each node u with capacity $c(u)$ has at most $c(u)$ robots on it. Subsequently the robots must terminate the algorithm.

Problem Statement: Byzantine Dispersion on a Capacitated Graph: Given k robots, up to f of which are Byzantine, initially placed arbitrarily on a capacitated graph of n nodes, the non-Byzantine robots must re-position themselves autonomously to reach a configuration where each node u with capacity $c(u)$ has at most $c(u)$ non-Byzantine robots on it. Subsequently the robots must terminate the algorithm.

1.2 Useful Procedures from Other Papers

We utilize two procedures from earlier literature. The first is the routine Explorer-Pebble, called EMT in [7], which can be used by at least two robots to construct a map of the graph, even when n is unknown. The procedure takes $O(n^3)$ rounds.

The second procedure we utilize, the universal exploration sequence (UXS) of [20], is one that allows a robot to visit all nodes of a graph of size at most n, when n is given as an input parameter. For any arbitrary graph, there exists a universal exploration sequence such that this procedure takes $O(n^5 \log n)$ time. To allow for various run times depending on prior knowledge of the graph (e.g., $O(d^2 n^3 \log n)$ for a d-regular graph [24]) or for how much memory a robot has, we simply say that the procedure takes $X(n)$ rounds.

1.3 Our Contributions

We make several contributions in this paper. First, we formalized the the problems of dispersion and Byzantine dispersion in the capacitated setting in this section.

In Section 2, we show how the use of a trusted shepherd to solve Byzantine dispersion leads to better guarantees. We first develop an algorithm that solves Byzantine dispersion on capacitated graphs in $O(X(n) + n^3)$ rounds and tolerates up to $\lfloor (k-1)/2 - 1 \rfloor$ strong Byzantine robots. This algorithm requires the shepherd to know the values of n and k. We then show how to replace the assumption that the shepherd knows the value of n with the assumption that the shepherd knows the value of f and develop an algorithm to solve Byzantine dispersion on capacitated graphs in $O(X(n) + n^3)$ rounds that tolerates up to $(k-1)/3 - 1$ strong Byzantine robots. Finally, we develop an algorithm that replaces the use of a trusted shepherd with the assumption that the total capacity of the graph is $\geq cf + k - f$, where c is the number of non-zero-capacity nodes. This algorithm solves Byzantine dispersion in $X(n)$ rounds, tolerates up to $k - 1$ strong Byzantine robots, and only requires the robots to know the value of n, unlike the previous algorithms. However, *all* robots must know the value of n.

Note that an algorithm for dispersion on uncapacitated graphs which includes map creation (e.g., [16, 17]) can be extended to capacitated graphs; simply add the capacities of nodes to the map as they are discovered. Thus, our work can immediately be compared with prior algorithms, both in terms of time and Byzantine tolerance. We include details in a table below.

Finally, in Section 3, we show that a shepherd aids in dispersion on a capacitated graph even without Byzantine robots. In this setting we use the shepherd's memory to store a map of the whole graph. We explain the technical challenges that render capacitated non-Byzantine dispersion a non-trivial task. We then develop a wrapper algorithm that (given any algorithm $\mathcal{A}$ for dispersion on uncapacitated graphs in $T_{\mathcal{A}}$ rounds and with $M_{\mathcal{A}}$ bits of memory per robot) solves dispersion on capacitated graphs in time $O(T_{\mathcal{A}} + m)$ with a requirement of $O(M_{\mathcal{A}})$ bits of memory for the non-shepherd robots and $O(M_{\mathcal{A}} + m \log k)$ bits of memory for the shepherd robot. We note that this wrapper algorithm works when $k > n$ and the values of $T_{\mathcal{A}}$, n, k, and any global knowledge parameters needed to run $\mathcal{A}$ are known to all robots.

We also develop algorithms that handle all values of k with less global knowledge. If only the shepherd knows n and no robot knows k, we give an algorithm using UXS that takes $O((X(n) + n^3 + m)$ time and $O(M_x + m \log(nk))$ bits of memory (where M_x is the memory to construct and use a UXS) for the shepherd and $O(\log k)$ bits of memory for the other robots. If no robot knows n or k, we show a $O(n^3 + m)$ time algorithm that assumes the shepherd is initially co-located with at least one other robot, the shepherd has $O(m \log(nk))$ bits of memory, and the remaining robots have $O(\log k)$ bits of memory.

We note that all our algorithms are deterministic in nature.

1.4 Comparison with Related Work

The problem of dispersion of mobile robots on graphs was originally introduced in [3]. Subsequent work [11, 12, 14, 23] in the synchronous system focused on reducing the time-memory trade-offs to solve the problem. The current best known algorithm is that of [14].

The problem has been extended to the asynchronous system [11, 14], dynamic graphs [1, 13], and randomized algorithms [6, 19, 21] among others. Here we consider robots with Byzantine faults [15, 16, 18].

In order to compare our work with prior related work, we first set our results in the same context. That is, we make the same assumption as prior work [16, 17] that $k = n$ (observe that in this case knowledge of n and knowledge of k are one and the same). Note that [17] contains the strongest results, and those results are what we compare ours with. In particular, we wish to highlight 3 of our results in this context. The algorithm in [17] handles up to $n-1$ weak Byzantine robots, where robots start from an arbitrary configuration, with the assumption that the quotient graph of the input graph is isomorphic to the input graph. We give an algorithm (Theorem 2.4) that handles up to $n-1$ *strong* Byzantine robots where robots start from an arbitrary configuration with the assumption that the total capacity of the input graph is $\geq cf + n - f$, where c is the number of non-zero capacity nodes. They have multiple results that do not require a restriction on the input graph and can handle robots starting from an arbitrary configuration. However, (i) their algorithms take *asymptotically longer* than our algorithm (Theorem 2.1), (ii) our algorithm has *strong Byzantine fault tolerance* that is similar but slightly less than the best weak Byzantine fault tolerance of their algorithms, and (iii) when compared with their algorithm that handles strong Byzantine robots, ours *requires one less parameter* to be known (i.e., f) and runs exponentially faster. A comparison of a subset of our results with prior work may be found in Table 1. We achieve these improvements through use of a "trusted shepherd".

To the best of our knowledge, no non-trivial time lower bounds are known for dispersion or Byzantine dispersion and a trivial lower bound of $\Omega(n)$ rounds holds.

2 THE POWER OF A SHEPHERD IN THE LAND OF BYZANTINE ROBOTS

2.1 Algorithm & Analysis

In this section, we present an algorithm that utilizes a trusted shepherd to allow robots to solve Byzantine dispersion on a capacitated graph, tolerating up to $\lfloor (k-1)/2 \rfloor$ strong Byzantine robots.

Brief Description. The algorithm is similar in structure to the algorithms from [16], i.e., it can be broken down into three stages, (i) gathering, (ii) map creation, and (iii) dispersion. However, we utilize the shepherd to execute these phases differently. In stage 1, the shepherd uses a UXS to find and gather the remaining robots. In stage 2, the gathered robots participate in an Explorer-Pebble routine, where the shepherd acts as the explorer and the remaining robots act as the pebble, to construct a map of the graph. Finally, in stage 3, the shepherd leads the other robots to find nodes to settle down at, before settling down itself.

The following theorem captures the properties of the algorithm. Due to space constraints, the detailed description of the algorithm and the proof of the theorem may be found in the full version [10].

Theorem 2.1. *There exists an algorithm that allows k robots, up to f of which are strong Byzantine robots, that are initially arbitrarily located on an n node capacitated graph to solve Byzantine dispersion in $O(X(n) + n^3)$ rounds when $f < \lfloor (k-1)/2 \rfloor$. Furthermore, k and n must be known to the shepherd.*

Notice that if robots are initially gathered, then they only need to run stage 2 and stage 3 of the algorithm, resulting in a faster run time. This is also an improvement on the total memory requirements of robots compared with [16]; here only one robot, the shepherd, creates and stores the map compared with all robots creating and storing maps in [16].

Corollary 2.2. *There exists an algorithm that allows k robots, up to f of which are strong Byzantine robots, to solve Byzantine dispersion on an n node graph in $O(n^3)$ rounds when $f < \lfloor (k-1)/2 \rfloor$ and all robots are initially gathered at the same node. Furthermore, k and n must be known to the shepherd.*

Table 1: Comparison of our results and previous results for Byzantine dispersion of k robots on an n node capacitated graph in the presence of at most f Byzantine robots. Note that previous results assume $k = n$ (so knowledge of n implies knowledge of k), whereas our results are for any k robots. In order to accurately compare our results with previous work, one should substitute $k = n$ in our results. Note that Λ_{good} is the length of the largest ID among non-Byzantine robots, Λ_{all} is the length of the largest ID among all robots, and $X(n)$ is the number of rounds required to explore any graph of n nodes.

Paper	Running Time (in rounds)	Starting Configuration	Byzantine Tolerance	Handles Strong Byzantine Robots	Required Knowledge
[16, 17]†*	$polynomial(n)$	Arbitrary	$n-1$	No	n
This paper (Theorem 2.4)†	$X(n)$	**Arbitrary**	$k-1$	**Yes**	n
[16, 17]†✠	$O(n^4\|\Lambda_{good}\|X(n))$	Arbitrary	$\lfloor n/2-1 \rfloor$	No	n
[16, 17]†◇	$O((f+\|\Lambda_{all}\|)X(n))$	Arbitrary	$O(\sqrt{n})$	No	n
[16, 17]†	$exponential(n)$	Arbitrary	$\lfloor n/4-1 \rfloor$	Yes	n and f
This paper (Theorem 2.1)ℵ	$O(X(n)+n^3)$	**Arbitrary**	$\lfloor (k-1)/2-1 \rfloor$	**Yes**	**n and k**
[16, 17]†	$O(n^4)$	Gathered	$\lfloor n/2-1 \rfloor$	No	n
[16, 17]†	$O(n^3)$	Gathered	$\lfloor n/3-1 \rfloor$	No	n
[16, 17]†	$O(n^3)$	Gathered	$\lfloor n/4-1 \rfloor$	Yes	n
This paper (Corollary 2.2)	$O(n^3)$	**Gathered**	$\lfloor (k-1)/2-1 \rfloor$	**Yes**	**n and k**

†This result assumes $k = n$.
*This result holds only for those graphs where the quotient graph is isomorphic to the original graph.
†This result holds only for those graphs where the total capacity of the graph $\geq cf + k - f$, where c is the number of non-zero capacity nodes.
✠Since $|\Lambda_{good}| = O(\log n)$ and $X(n) = \tilde{O}(n^5)$ (see [2, 24]), $O(n^4|\Lambda_{good}|X(n)) = \tilde{O}(n^9)$.
◇Since $|\Lambda_{all}| = O(\log n)$, $f = O(\sqrt{n})$, and $X(n) = \tilde{O}(n^5)$ (see [2, 24]), $O((f + |\Lambda_{good}|)X(n)) = \tilde{O}(n^5\sqrt{n})$.
ℵSince $X(n) = \tilde{O}(n^5)$ (see [2, 24]), $O(X(n) + n^3) = \tilde{O}(n^5)$.

2.2 Replacing the Shepherd's Knowledge Requirement of n with f

In this section, we design an algorithm that substitutes the shepherd's knowledge requirement of n with that of f. However, the tolerance of the algorithm to Byzantine robots is reduced, i.e., the algorithm can only handle $f < (k-1)/3$ strong Byzantine robots. Due to space constraints, the detailed algorithm and the proof of the theorem can be found in the full version [10].

Brief Description. This is a three stage process. In stage one, all non-shepherd robots perform UXSes with input parameter 2^i for increasing values of $i = 1, 2, 3, \ldots$ until they find the shepherd. The shepherd waits until at least $k - f - 1$ other robots find it and then moves to stage two. In stage two, the shepherd and the gathered robots[3] construct the map of the graph using an explorer-pebble routine, resulting in the shepherd knowing the value of n. The final stage consists of the shepherd waiting at a node for a sufficient amount of time to allow for any remaining non-Byzantine robots find it, then settling the robots and itself as before.

Theorem 2.3. *There exists an algorithm that allows k robots, up to f of which are strong Byzantine robots, that are initially arbitrarily located on an n node capacitated graph to solve Byzantine dispersion in $O(X(n) + n^3)$ rounds when $f < (k-1)/3$. Furthermore, k and f must be known to the shepherd.*

[3]Since there are at least $k - f - 1$, a majority are non-Byzantine.

2.3 Replacing the Shepherd with an Input Condition

Here we give an algorithm to solve Byzantine dispersion without the use of a shepherd, as long as input conditions are met. Due to space constraints, the detailed algorithm and the proof of the theorem can be found in the full version [10].

Brief Description. Each robot R explores the graph according to the universal exploration sequence with input parameter n for $X(n)$ rounds. At each timestep, at each node, the robots present check its remaining capacity and then that many robots settle there while the remainder (if any) keep exploring. After $X(n)$ rounds are over, the robot terminates the algorithm.

Theorem 2.4. *There exists an algorithm that allows k robots who all know n, up to f of which are strong Byzantine robots, that are initially arbitrarily located on an n node capacitated graph, to solve Byzantine dispersion in $X(n)$ rounds when $f < k$ and the total capacitance of the graph is $\geq cf + k - f$, where c is the number of non-zero capacity nodes.*

3 THE USE OF A SHEPHERD IN THE CAPACITATED MODEL WITH NO BYZANTINE ROBOTS

In this section, we show how a trusted shepherd can be beneficial even in the absence of Byzantine robots. In Section 3.1, we present a wrapper algorithm that allows one to solve dispersion on a capacitated graph using any pre-existing dispersion algorithms for an uncapacitated graph. We make the assumption that $k \geq n$ for this algorithm, which is natural (e.g., consider cars navigating a city in search of parking where roads are edges and garages are nodes). In Section 3.2, we show how to remove this assumption at the expense of a possibly larger run time and possibly more memory required by the shepherd.

Motivation and Technical Challenges:

A typical dispersion algorithm would explore the graph by settling robots on vacant nodes as they are encountered, and using those settled robots as signposts for unsettled robots. That process will not work for the capacitated version: Suppose we have to disperse k robots on an n node graph where only two nodes have non-zero capacity. We have a contradiction once the first but not the second non-zero-capacity node is found. The robots should settle there (and leave their zero-capacity nodes unsettled), but on the other hand the robots must remain on the zero-capacity nodes to facilitate exploring the rest of the graph and finding the second non-zero capacity node.

The situation becomes even more difficult if the number of robots k is less than the number of nodes n, in which case the robots may not even have enough memory collectively to store a map of the graph. Our algorithm successfully overcomes both of these issues.

Notes and Assumptions: We note that pre-existing algorithms for dispersion often assume that either $k = n$ or $k < n$, but those algorithms can be easily converted into algorithms for $k \geq n$ when the value of $\lceil k/n \rceil$ is known to the robots: Instead of assuming only 1 robot may settle at a node, they may assume that up to $\lceil k/n \rceil$ robots may settle at each node and act appropriately. An additional $O(\log(k/n))$ bits of memory per robot is needed, however that is subsumed by the $O(\log k)$ bits for each robot to store its own unique ID.

We assume that all robots have access to a dispersion algorithm $\mathcal{A}$, know its run time $T_{\mathcal{A}}$ and have enough memory $M_{\mathcal{A}}$ to run $\mathcal{A}$. We require that all robots know n and k and also whatever assumptions are needed in order to run $\mathcal{A}$.

3.1 Handling $k \geq n$ Robots

As discussed above, the primary challenges for dispersion across a capacitated graph are creating a map of the graph and then determining where each robot should settle. We use a shepherd to deal with both difficulties.

Brief Description. Our wrapper algorithm consists of three phases. In phase one, all robots run an already existing dispersion algorithm $\mathcal{A}$, treating the graph as uncapacitated. In phase two, the shepherd performs a depth first search (DFS) of the graph to construct a map of the graph using the temporarily settled robots to differentiate nodes. In phase three, the shepherd first collects all the robots using a DFS as before, then performs a second DFS allocating robots to nodes subject to capacity constraints. When the shepherd allots a robot to a node, that robot terminates the algorithm. Once all robots (including the shepherd) are allotted to nodes, the shepherd terminates.

Theorem 3.1, given below, captures the properties of the algorithm. Due to space constraints, the detailed version of the algorithm as well as the proof of the theorem may be found in the full version [10].

Theorem 3.1. *Assume that k robots have access to an algorithm $\mathcal{A}$ that solves dispersion on an uncapacitated graph of n nodes in time $T_{\mathcal{A}}$ and requires each robot to have $M_{\mathcal{A}}$ bits of memory. If $k \geq n$ and all the robots know the value of $T_{\mathcal{A}}$, n, k, and whatever other global knowledge is required to run $\mathcal{A}$, then there exists an algorithm to solve dispersion of k robots on an n node capacitated graph in time $O(T_{\mathcal{A}} + m)$, where m is the number of edges of the graph, with a memory requirement of $O(M_{\mathcal{A}} + m \log k)$ bits of memory for a shepherd robot and $O(M_{\mathcal{A}})$ bits of memory for the remaining $k - 1$ robots.*

The current best known algorithm for dispersion on an uncapacitated graph is that of [14], which allows k robots initially arbitrarily located on an n node graph (where $k \leq n$) to achieve dispersion in $O(\min\{m, k\Delta\})$ time, where Δ is the maximum degree of the graph and m is the number of edges, and requires each robot to have $O(\log(k + \Delta))$ bits of memory. From the previous theorem and our discussion just prior to Section 3.1 on how to convert dispersion algorithms on uncapacitated graphs that work for $k \leq n$ to dispersion algorithms on uncapacitated graphs that work for $k \geq n$, we have the following corollary.

Corollary 3.2. *Assume that k robots have access to the dispersion algorithm $\mathcal{A}$ from [14] for uncapacitated graphs. If $k \geq n$ and all the robots know the values of $O(\min\{m, k\Delta\})$ (where m is the number of edges of the graph and Δ is the maximum degree of the graph), n, k, and the global knowledge requirements of algorithm $\mathcal{A}$, then there exists an algorithm to solve dispersion of k robots on an n node capacitated graph in time $O(m)$ with a memory requirement of $O(m \log k)$ bits of memory for the shepherd robot and $O(\log(k + \Delta))$ bits of memory for the remaining $k - 1$ robots.*

3.2 Handling Any Value of k Robots

The algorithm in Section 3.1 assumed that the number of robots k was greater than or equal to the number of nodes n. Here, we describe a modification of the preceding algorithm to deal with scenarios where k is less than n or k is unknown. To handle these situations, we use a UXS or the Explorer-Pebble routine, both described in Section 1.2, rather than a pre-existing dispersion algorithm. In the below descriptions, we use M_x to denote the memory required by a robot to construct and use a UXS.

Brief Description. First, the shepherd runs a UXS with input parameter n until it finds another robot. Next, these two robots perform the Explorer-Pebble routine where the shepherd acts as the explorer to construct a map of the graph which is stored with the shepherd. The shepherd learns the value of k while constructing the map by noting how many robots are at each node. After mapping, the shepherd runs phase three of the algorithm from Section 3.1 to collect and then disperse the robots.

Note that in this algorithm, only the shepherd needs to know the value of n, i.e., n does not need to be global knowledge, and k can be unknown. Also, the non-shepherd robots only need $O(\log k)$ bits of memory each to store their unique IDs. Since k may be less than n, the shepherd's memory required for the map is $O(m \log(nk))$.

Theorem 3.3. *There exists an algorithm that allows k robots that are initially arbitrarily located on an n node capacitated graph with m edges to solve dispersion in time $O(X(n) + n^3 + m)$, requires the shepherd to have $O(M_X + m \log(nk))$ bits of memory, where M_X is the memory required to construct and use a UXS and $X(n)$ is the running time of that procedure, and the remaining robots to have $O(\log k)$ bits of memory. Only the shepherd needs to know the value of n.*

Notice that the running time may be large due to the UXS. If we assume that all robots are initially gathered, or that the starting configuration is such that at least one other robot starts on the same node as the shepherd, then we may skip the use of the UXS and directly move to the use of the Explorer-Pebble routine to construct the map of the graph. In this scenario, none of the robots need to know either the value of n or k.

Theorem 3.4. *There exists an algorithm that allows k robots that are initially located such that the shepherd and at least one robot initially occupy the same node on an n node capacitated graph with m edges to solve dispersion in time $O(n^3 + m)$ and requires the shepherd to have $O(m \log(nk))$ bits of memory and the remaining robots to have $O(\log k)$ bits of memory. None of the robots need to know the value of n or k.*

4 CONCLUSION

This work suggests several new lines of future research. Here we analyzed dispersion on graphs where nodes have different capacities; what if the robots have different needs as well? This is a model for when some robots need more of a given resource (e.g., space to park, energy at a charging station, etc.). Coupled with capacitated graphs, this would be a more realistic analogue to many real-world situations.

Another line of research is exploring the utility of a trusted shepherd in other variants of the traditional models for dispersion and Byzantine dispersion. For instance, when dealing with dynamic graphs, could a slightly more powerful robot aid the algorithm designer? What power would be most helpful?

A third line of research is to see whether the algorithms in this paper may be adapted to an asynchronous setting. Some of the algorithms, especially those for Byzantine dispersion, may appear to be asynchronous in nature as they are event driven (as opposed to time dependent) and the shepherd is the robot that tells other robots to terminate the algorithm. However, a key component of those algorithms is the communication inherent in each round. If there were no rounds, then it becomes difficult to tell whether all robots co-located at a node had the chance to communicate with one another. It especially becomes difficult to differentiate between a relatively "slow" robot and one that is acting in a Byzantine manner by refusing to communicate. Perhaps the additional power of the trusted shepherd could be used to overcome these challenges.

REFERENCES

[1] Ankush Agarwalla, John Augustine, William K. Moses Jr., Madhav Sankar K., and Arvind Krishna Sridhar. 2018. Deterministic Dispersion of Mobile Robots in Dynamic Rings. In *ICDCN*. ACM, 19:1–19:4.

[2] Romas Aleliunas, Richard M. Karp, Richard J. Lipton, László Lovász, and Charles Rackoff. 1979. Random walks, universal traversal sequences, and the complexity of maze problems. In *SFCS*. IEEE Computer Society, 218–223.

[3] John Augustine and William K. Moses Jr. 2018. Dispersion of Mobile Robots: A Study of Memory-Time Trade-offs. In *ICDCN*. ACM, 1:1–1:10.

[4] Mark Cieliebak and Giuseppe Prencipe. 2002. Gathering Autonomous Mobile Robots. In *SIROCCO*. 57–72.

[5] Reuven Cohen, Pierre Fraigniaud, David Ilcinkas, Amos Korman, and David Peleg. 2008. Label-guided graph exploration by a finite automaton. *TALG* 4, 4 (2008), 42.

[6] Archak Das, Kaustav Bose, and Buddhadeb Sau. 2021. Memory Optimal Dispersion by Anonymous Mobile Robots. In *CALDAM*. Springer, 426–439.

[7] Yoann Dieudonné, Andrzej Pelc, and David Peleg. 2014. Gathering despite mischief. *TALG* 11, 1 (2014), 1–28.

[8] Yotam Elor and Alfred M Bruckstein. 2011. Uniform multi-agent deployment on a ring. *Theoretical Computer Science* 412, 8-10 (2011), 783–795.

[9] Duncan Graham-Rowe. [n. d.]. Robotic Weather Planes. *MIT Technology Review* ([n. d.]). https://www.technologyreview.com/2008/12/16/32928/robotic-weather-planes/

[10] William K. Moses Jr. and Amanda Redlich. 2023. Dispersion, Capacitated Nodes, and the Power of a Trusted Shepherd. *CoRR* abs/2311.01511 (2023). https://arxiv.org/abs/2311.01511

[11] Ajay D. Kshemkalyani and Faizan Ali. 2019. Efficient Dispersion of Mobile Robots on Graphs. In *ICDCN*. 218–227.

[12] Ajay D. Kshemkalyani, Anisur Rahaman Molla, and Gokarna Sharma. 2019. Fast Dispersion of Mobile Robots on Arbitrary Graphs. In *ALGOSENSORS*. Springer, 23–40.

[13] Ajay D. Kshemkalyani, Anisur Rahaman Molla, and Gokarna Sharma. 2020. Efficient Dispersion of Mobile Robots on Dynamic Graphs. In *ICDCS*. IEEE, 732–742.

[14] Ajay D. Kshemkalyani and Gokarna Sharma. 2022. Near-Optimal Dispersion on Arbitrary Anonymous Graphs. In *OPODIS*.

[15] Anisur Rahaman Molla, Kaushik Mondal, and William K. Moses Jr. 2020. Efficient Dispersion on an Anonymous Ring in the Presence of Weak Byzantine Robots. In *ALGOSENSORS*. 154–169.

[16] Anisur Rahaman Molla, Kaushik Mondal, and William K. Moses Jr. 2021. Byzantine Dispersion on Graphs. In *IPDPS*. IEEE, 942–951.

[17] Anisur Rahaman Molla, Kaushik Mondal, and William K Moses Jr. 2021. Byzantine Dispersion on Graphs. *arXiv preprint arXiv:2102.07528* (2021).

[18] Anisur Rahaman Molla, Kaushik Mondal, and William K. Moses Jr. 2021. Optimal Dispersion on an Anonymous Ring in the Presence of Weak Byzantine Robots. *TCS* 887 (2021), 111–121.

[19] Anisur Rahaman Molla and William K. Moses Jr. 2019. Dispersion of Mobile Robots: The Power of Randomness. In *TAMC*. Springer, 481–500.

[20] Omer Reingold. 2008. Undirected connectivity in log-space. *JACM* 55, 4 (2008), 1–24.

[21] Nicolás Rivera, Thomas Sauerwald, Alexandre Stauffer, and John Sylvester. 2019. The Dispersion Time of Random Walks on Finite Graphs. In *SPAA*. 103–113.

[22] Yara Rizk, Mariette Awad, and Edward W. Tunstel. 2019. Cooperative heterogeneous multi-robot systems: A survey. *ACM Computing Surveys (CSUR)* 52, 2 (2019), 1–31.

[23] Takahiro Shintaku, Yuichi Sudo, Hirotsugu Kakugawa, and Toshimitsu Masuzawa. 2020. Efficient Dispersion of Mobile Agents without Global Knowledge. In *SSS*. Springer, 280–294.

[24] Amnon Ta-Shma and Uri Zwick. 2014. Deterministic Rendezvous, Treasure Hunts, and Strongly Universal Exploration Sequences. *ACM Trans. Algorithms* 10, 3 (2014), 12:1–12:15.