4-21-2015

Searching and Sorting Algorithms

Richard Alexander Beckwith Johnson

SEARCHING AND SORTING ALGORITHMS

by

Richard A. B. Johnson

A Dissertation

Submitted in Partial Fulfillment of the

Requirements for the Degree of

Doctor of Philosophy

Major: Mathematical Sciences

The University of Memphis

May 2015

ACKNOWLEDGEMENTS

First, and above all, I would like to thank my supervisor Professor Béla Bollobás. Without him I would never have had the opportunity to even embark on this PhD in the first place. His continual support, both academic and extra-curricular, over the past four years has allowed me to visit many other institutions, introduced me to a huge variety of colleagues to collaborate with and interesting problems to work on. The results contained in this dissertation are all due to his relationships formed from his introductions, and represent a small subset of the things he has helped me to achieve. I would also like to thank Professor Paul Balister who has always provided me with an open ear regarding any mathematical questions I had, frequently helping me to see to the heart of a problem almost instantly.

I am also very indebted to Gabriella Bollobás for her endless hospitality. She has always welcomed all of us into her home, made us feel very comfortable and looked after us with any problems we have. She has helped me to grow as an individual as much as Béla has as a mathematician, and I am very grateful for that. I am also very grateful to Tricia Simmons, who began helping me even before I arrived in Memphis. She has been a pillar of support to all of Béla's students over the years, and we all know how special she is to all of us.

One of the most wonderful things about the past four years has been joining the family of Béla's students, both current and past. The twin groups of the Memphis and Cambridge students work very closely together, and I have enjoyed getting to know, work with and enjoy the company of, each and every one of them. I would like to individually thank two in particular, Tomas Juškevičius and Michał Przykucki. They have always shown me unending support, sympathised with my difficulties and even occasionally offered useful advice. Away from the department I would like to thank Stacey, Steph, Jess, Mansi and Kirstin for their support.

Moving to a foreign country away from your family and friends can be a very stressful undertaking, and you helped to significantly reduce the disruption I felt.

I am very lucky to have discovered my friends in the Hash House Harriers, who have always been more than willing to drop whatever they were doing in search of a fun new area of town to explore, or just to share in a few pints. Above all it was through them that I met Cailin, who has been amazing through the final stages of this PhD. She makes me smile every day, and I look forward enormously to her accompanying me on my next adventure.

Finally I owe a debt of gratitude, as ever, to my long-suffering parents and sister, who have had to endure another four years of my frequent needs for places to stay in England, financial assistance and moral support. I promise this is the last of my overseas adventures, and I look forward to finally coming home and spending more time with you all.

ABSTRACT

Johnson, Richard A. B. Ph.D. The University of Memphis. May 2015.
Searching and Sorting Algorithms. Major Professor: Béla Bollobás, Ph.D.

This dissertation analyses two combinatorial questions that involve algorithmic solutions. First we consider the Robber Locating Game, a pursuit-evasion game introduced by Seager in 2012. This game is a variant of the renowned Cops and Robbers game; in this variant the robber does not disclose his location to the cop, and her aim is merely to locate rather than capture him. Although he moves around the graph as normal on his turns, on her turns she picks any vertex freely and asks how far he is from her probed vertex. We call a graph *locatable* if there is a possible cop strategy that will always locate the robber in finitely many moves, and *non-locatable* otherwise.

In this dissertation we consider how much subdivision of a graph is necessary to make it locatable, establishing exact bounds in the case of complete and complete bipartite graphs, and a general $n/2 + 1$ bound for all finite graphs. We also consider subdividing infinite graphs, exhibiting a sufficient subdivision function for the cases where subdividing them can make them locatable. Finally we close with a series of results about the game, including the relationship between locatability number and maximum degree and showing that every locatable graph is 4-colourable.

In the second part we consider how a user can determine the ordering of a well-ordered set of elements, when he initially does not know the ordering but is given a scale. This scale takes k elements and returns the $t_1^{\text{st}}, t_2^{\text{nd}}, \ldots, t_s^{\text{th}}$ of them according to this ordering. We show that he cannot determine the complete ordering, since he cannot order the initial and final segments. Apart from this restriction we outline algorithms to enable the user to determine the ordering in both the online and offline cases. We show that in the online case he can determine the ordering in $O(n \log n)$ queries, and in the offline case in $O(n^{k-(t-1)})$ queries, which we show is the best possible order of the number of queries.

TABLE OF CONTENTS

LIST OF FIGURES

INTRODUCTION

This dissertation discusses two different combinatorial questions that both have algorithmic solutions. In the first part we analyse a variant of the classic Cops and Robbers problem, an example of a Pursuit and Evasion game. Pursuit and evasion games on graphs have been widely studied. The most standard example is the Cops and Robbers game, an instance of which is a graph G together with a fixed number of cops. The cops take up positions on vertices of G and a robber then starts on any unoccupied vertex. The cops and the robber take turns: the robber chooses either to remain at his current vertex or to move to any adjacent vertex, and then the cops simultaneously make moves of the same form. The game is played with perfect information, so that at any time each of the players knows the location of all others. The cops win if at any point one of them is at the same location as the robber. The cop number of a graph is the minimum number of cops required for the cops to have a winning strategy. Early results on this game include those obtained by Nowakowski and Winkler [9], who describe the graphs of cop number 1, and Aigner and Fromme [1], who show that every planar graph has cop number at most 3. Perhaps the most important open problem in this area is Meyniel's conjecture (see Frankl [5]), that the cop number of any n-vertex connected graph is at most $O(\sqrt{n})$. This has been shown to be true up to a $\log(n)$ factor for random graphs by Bollobás, Kun and Leader [2], following which Łuczak and Prałat improved the error term [8]. More recently, several variations on the game have been analysed by Clarke and Nowakowski [4].

Our aim in this dissertation is to study the Robber Locating game, introduced in a slightly different form by Seager [10], and further studied by Carraher, Choi, Delcourt, Erickson, and West [3]. In this game the robber initially occupies a vertex, without disclosing which it is to the cop. For ease of reading we shall refer to the cop as female and the robber as male. Each round consists of a move for the

robber, in which he either moves to an adjacent vertex or stays where he is, followed by a probe of any vertex by the cop. When the cop probes a vertex she is told the current distance to the robber. In this setting the cop is not on the graph herself, and can probe vertices without restriction; she wins if at any point she is able to determine the robber's current location. We say that a graph is *locatable* if a strategy allowing her to do so in finite time exists and *non-locatable* otherwise.

The default view of this game has two significant weaknesses. The first is that it seems that the cop will clearly win eventually with probability 1 on a finite graph against a robber who has no knowledge of her future moves, simply by probing random vertices until she hits the current location of the robber. The second is that no possible winning robber strategy can exist, since it must prescribe a starting vertex, and then any cop strategy that begins by probing that vertex would locate him. As such in this dissertation we open by presenting an alternative version of this game, in which the robber occupies a subset of the vertices, and the cop wins if she can reduce this subset to a singleton. As we shall show these games are so closely related that they are almost functionally equivalent from the cop's perspective, and hence we adopt this alternative view without losing touch with the default game. It does however allow us to define winning robber strategies, and to show that any non-locatable graph has a winning robber strategy. We follow this with a chapter in which we show some simple results about the Robber Locating Game, which are included mostly for completeness rather than because they are particularly insightful. The real content of this dissertation begins in Chapter 3.

In [3] Carraher, Choi, Delcourt, Erickson, and West analysed the Robber Locating Game with subdivisions in mind, and showed that subdividing any finite graph sufficiently many times yields a locatable graph. Given a graph G and a positive integer m, we write $G^{1/m}$ for the graph obtained by replacing each edge of G by a path of length m through new vertices. Each such path is called a *thread*,

and an *original vertex* in $G^{1/m}$ is a vertex that corresponds to a vertex of G. We denote the path between original vertices v_i and v_j by $v_i \cdots v_j$. The *span* of an original vertex consists of all the vertices at distance less than m from it; this set includes the vertices along the threads leaving that vertex, but not the far endpoints of those threads.

In [3] the authors proved a general sufficient bound on the amount of subdivision required to make a finite graph locatable, phrased partially in terms of the metric dimension of the graph. The notion of metric dimension was introduced independently by Slater [12], and by Harary and Melter [7]. The metric dimension of G, denoted $\mu(G)$, is the size of the smallest set S of vertices such that for every $x, y \in V(G)$ with $x \neq y$ there is some $z \in S$ with $d(x, z) \neq d(y, z)$. In [3] the authors proved that $G^{1/m}$ is locatable provided

$$m \geqslant \min\{n(G), 1 + \max\{\mu(G) + 2^{\mu(G)}, \Delta(G)\}\}$$

Further they showed that for a complete bipartite graph $K_{a,b}$, $m \geqslant \max\{a, b\}$ is sufficient for $K_{a,b}^{1/m}$ to be locatable. They asked if this was necessary – in Chapter 3 we show that in fact $m \geqslant (\min\{a, b\} - 1)$ is necessary and sufficient if $\min\{a, b\} \geqslant 4$, and $m \geqslant \min\{a, b\}$ is necessary and sufficient if $\min\{a, b\} \leqslant 3$, classifying all subdivided complete bipartite graphs. They also conjecture that their bound is tight for complete graphs, i.e. that $K_n^{1/m}$ is locatable if and only if $m \geqslant n$. We show that in fact, except for a few small values of n, the actual threshold is $n/2$. We then extend this to show that, for all finite graphs, subdividing by $n/2 + 1$ is sufficient to make a graph locatable, a bound that is tight for small complete graphs.

We then extend this approach in Chapter 4 to consider subdividing infinite graphs. We show that any infinite graph with any vertices of infinite degree, or infinitely many components, is not locatable and since these properties are preserved under subdivisions, no subdivision of such a graph can be locatable.

However, given an infinite graph not having one of those properties, we show that there exists a subdivision function that makes it locatable. We note that, although this function is not necessarily uniform, in the case of regular infinite graphs it is uniform, and provided that the underlying graph has bounded degree then uniformly subdividing by the maximum also gives a valid uniform subdivision that makes the graph locatable.

We then move to more general results about the Robber Locating Game. In Chapter 5 we ask if it is possible to bound the time that the cop will take to locate the robber on a locatable graph in terms of the maximum degree. Unsurprisingly (as we show in Chapter 4) the robber can always survive for $\log \Delta$ turns. We believe that this is the correct lower bound, and as evidence towards this we give a graph on which he can be located in $\log \Delta + \log(\log \Delta) + 2 = O(\log \Delta)$ turns. We also show that no upper bound is possible by exhibiting a family of graphs with bounded degree, which are locatable, on which the robber can survive for arbitratily long.

We close the section on the Robber Locating Game with two chapters inspired directly by Seager's initial paper, [10]. In this paper she asked if a forbidden subgraph characterisation of locatable graphs was possible, i.e. if it was possible to give a family of graphs $\mathcal{H}$ such that the locatable graphs are exactly those not containing a member of $\mathcal{H}$ as a subgraph. In Chapter 6 we show that this is not possible by exhibiting a locatable graph that contains an unlocatable induced subgraph. Finally in Chapter 7 we consider the relationship between locatability and colourability, and show that all locatable graphs are 4-colourable. We also show that this is tight by exhibiting a locatable graph that is not 3-colourable. In both of these final two chapters we include the *no-backtrack* condition that Seager originally included: this states that the robber cannot move to the vertex that the cop most recently probed. Although her initial paper included this, neither the followup by Carraher, Choi, Delcourt, Erickson, and West [3] or Seager [11] do, and we consider

it to be a slightly unnatural condition. This is included in Chapter 6 since the question posed by Seager included it, and in Chapter 7 we also include a note on how the results change if this condition is dropped.

In the second part of this dissertation we consider an unrelated problem about a sorting algorithm. Consider a set of ordered elements $\{x_1, \ldots, x_n\}$ and a user who does not know the ordering but wishes to establish it. There are many similar such problems, often phrased in terms of a number of coins with distinct weights, and a person who wishes to sort them. The user is allowed access to a weighing scale that accepts k of them as input and returns a subset of size s containing the $t_1^{\text{st}}, t_2^{\text{nd}}, \ldots, t_s^{\text{th}}$ elements. We would call such a scale a $(k, t_1, \ldots, t_s)$ scale, and in the special case when $k = 2$ we call it a *binary scale*. We investigate what such a scale can tell the user about the ordering, and how quickly he can determine this information.

Clearly the user cannot discover the complete ordering, as he cannot determine the ordering of the first $t_1 - 1$ elements or the final $(n - t_s)$ elements. Additionally if the scale is symmetric (in the sense that the elements that it returns are symmetric around the midpoint of k i.e. $t_1 = k - t_s, t_2 = k - t_{s-1}$ etc) then the ordering cannot be fully determined for the remaining elements, as the results of any query would be the same if the ordering was reflected. Aside from these restrictions, everything else about the ordering can be determined, as we show. We consider both online algorithms, in which the user can decide which set to query next based on previous results, and offline algorithms in which he must specify all his queries initially before receiving any results. In the former case we achieve a general bound of $O(n \log n)$ queries, which was already well known in the binary case, but with a better leading constant for large k. In the offline case we establish two algorithms that can determine the ordering, both in $O(n^{k-(t-1)})$ queries, which is the best possible order. The former works by fixing a small set of reference elements and requesting all the

queries involving that set. From these results it recursively builds up the results of any query involving any subset of these reference elements, and thus establishes how to find the results of any query. From that point the online algorithms can be applied to determine the ordering. The second algorithm is more direct, and relies on establishing the adjacencies of elements in the ordering and hence deducing as much about it as is possible. We discuss both algorithms for the singleton output ($s = 1$) and multiple output ($s > 1$) cases, arguing in favour of using the adjacency based algorithm especially in the multiple output case.

The work on the Robber Locating Game is all joint work with Sebastian Koch of Cambridge, and additionally with John Haslegrave of Sheffield for Chapter 3. The work on the Scales algorithms is all joint with Gábor Mészáros of Central European University.

Part I

The Robber Locating Game

CHAPTER 1

FORMAL DESCRIPTION OF THE ROBBER LOCATING GAME

The original presentation of the game considers the robber as occupying a single vertex, whose location is unknown to the cop, and which she wishes to determine. He initially picks a vertex to start on, and then at each round proceeds as follows. First she picks a vertex to *probe*, and he tells her his distance from the probed vertex. If following this she can work out his location then she wins the game. Otherwise he can then move to an adjacent vertex or remain still. This concludes a round of the game. He wins the game if he can evade being located indefinitely. There is one additional complication, which was included by Seager in her original paper [10], the *no backtrack condition*, which states that the robber cannot move to the vertex just probed by the cop. This was however not included in the later papers by Carraher, Choi, Delcourt, Erickson, and West [3] or Seager's followup paper [11]. We shall in general not include this condition, to simplify the description of the game and to follow what appears to be the current popular consensus. The exceptions to this are Chapters 6 and 7, where we need it in order to answer the questions posed by Seager exactly. However otherwise it does not feature in our analysis.

Formally a strategy for the robber specifies where he will begin, and on each turn where he will move (which may depend on the previous probe vertices and answers he has given the cop). Equally a strategy for the cop consists of an initial choice of vertex to probe, and then subsequent choices depending on the answers that the robber gives. Given strategies for both players either the cop locates the robber after finite time, or she does not – in the former case we say that the cop wins, and in the latter we say that the robber wins if both players use those particular strategies.

We say that a strategy for the cop is a *winning strategy* if it beats all possible

robber strategies. This is a property of the strategy, which is again a property of the underlying graph. Hence we can define a graph as *locatable* if there exists a winning strategy for the cop, or *non-locatable* otherwise. Winning strategies for the robber are not possible, since a robber strategy formally includes his choice of starting vertex, and for any choice of starting vertex there is a trivial cop strategy (repeatedly probing that vertex on each turn) that locate him on the first probe.

Further if the cop is allowed to probe randomly then the game is essentially trivial, since picking probe vertices uniformly at random from the vertex set is a simple strategy that works almost surely on any finite graph. In order to counteract both these problems, we present an alternative version of the game, which we call the Power Set version, that has the following advantages:

1. it forbids random strategies, clearly defining the criteria for winning strategies.
2. it gives rise to an algorithm that determines if a graph is locatable or non-locatable in this version of the game in finite time on any finite graph.
3. in the absence of a winning strategy for the cop it gives a winning strategy for the robber.
4. it better captures how the game is analysed in practice.

As we shall see later there is an obvious coupling between the two versions of the game, which gives rise to a natural bijection between winning strategies for the cop in each. Hence a graph that's locatable for the Power Set version of the game is also locatable for the default Robber Locating game, and vice versa. Thus locatability is preserved as you move between the different versions. Hence when we describe a graph as locatable, or non-locatable, we do not need to specify which version of the game we are talking about.

In our alternative version we allow the robber to occupy a subset of the vertex set rather than a single vertex. This would correspond to the set of vertices that the

cop knows he must be contained within in the original game. If it is ever a singleton vertex then that corresponds to the cop locating him, and we say that the cop wins. Otherwise if he can avoid ever being in a singleton then we say he wins, since this corresponds to there always being some uncertainty as to his position.

We use R_t to represent the set that he occupies after the t^{th} probe but before his t^{th} move, and M_t to represent the set of vertices that the robber occupies after his t^{th} move. The R_t are called the *response sets* and the M_t are called the *movement sets*, since the game plays as follows. Initially he occupies $M_0 = V$, the entire vertex set, since his start position could be any vertex in the graph. The cop then picks a probe vertex, and his response determines a subset of M_0, namely R_1, his first response set. If this is a singleton set then she has located him. Otherwise he can then move to the closed neighbourhood of R_1, which is thus the first movement set M_1. The game then continues as such for each subsequent probe.

It is straightforward to write recurrence relations for how the game progresses. Let the cop's probe at time t be denoted by p_t. The robber's response can either be described by a distance d_t or directly by the set R_t that is created by this response. For clarity, and to better establish the coupling with the original game, formally we define the game as him returning this distance and say R_t is determined by his response, although practically, with abuse of notation, in our later analysis we shall often shorten this to just say he responds with R_t. With this notation in mind, we can define how the game progresses as follows:

$$M_0 = V \tag{1.1}$$

$$R_t = \{v \in M_{t-1} : d(v, p_t) = d_t\} \tag{1.2}$$

$$M_t = R_t \cup \{v \in V : d(v, R_t) = 1\} \tag{1.3}$$

We can now give a more explicit definition of a robber strategy, since the robber only needs to consider the set of vertices that he was previously known to be in (L_{i-1}) and the probe vertex. Further, we can give an explicit description of what constitutes strategies for the cop and robber. Note that the game is effectively memoryless, the current state of the game is entirely encapsulated by the current value of M_t. Hence a cop strategy consists of specifying, for each possible value of $M_t \subset V$, which vertex to probe next, and a robber strategy consists of specifying, for each possible value of M_t and choice of probe vertex which distance to return. Denoting a cop strategy by S_C and a robber strategy by S_R, strategies are hence finite maps of the following forms:

$$S_C : \mathcal{P}(V) \to V \tag{1.4}$$

$$S_R : \mathcal{P}(V) \times V \to \mathbb{N} \tag{1.5}$$

With these definitions in mind, we shall now prove that this alternative version satisfies the Claims 1 - 4 above. The first is well illustrated by considering playing the game on a complete graph, K_n. Previously on each turn probing a vertex chosen uniformly at random gave each probe a uniform $1/n$ success probability. In this game however the robber can always return distance 1, and thus $R_t = V \setminus \{p_t\}$ at every time step. Provided $n \geqslant 3$ this never locates him, and so a random strategy no longer works.

The second claim we shall show explicitly in the following theorem, which outlines the algorithm.

Theorem 1. There is an algorithm that can determine for any graph G if G is locatable or non-locatable.

Proof. The algorithm works by creating an auxiliary graph on the state space of the game. We construct this auxiliary graph H as follows. The vertices represent the

state space of the game, so are subsets of the original vertex set. Hence $V(H) = \mathcal{P}(V(G))$. Our edges will represent the possible outcomes for probes by the cop. Given that the robber at some point is in set M if the cop probes vertex p then he can choose which distance he returns from $\{1, \ldots, \text{diam}(G)\}$. Let $L(M, p, d)$ represent the layer of set M at distance d from p, so

$$L(M, p, d) = \{v \in M : d(v, p) = d\} \tag{1.6}$$

There are two possible outcomes, either $|L(M, p, d)| \leq 1$, in which case the cop immediately wins, or $|L(M, p, d)| > 1$, in which case the robber could return this set and then move to its closed neighbourhood. Let us denote the closed neighbourhood of $L(M, p, d)$ by $N[L(M, p, d)]$. Then for each $d \in \{1, \ldots, \text{diam}(D)\}$ we construct an edge from M to either the empty set (which we interpret as a winning state for the cop) or $N[L(M, p, d)]$ accordingly. We would have a set of such edges for each $p \in V(G)$, one for each choice of probe vertex. We suggest that it would be a good idea to colour such edge sets according to the probed vertex, using $|V(G)|$ colours. Doing this for all $p \in V(G), M \subset V(G)$ gives the edges of H.

For the purpose of illustration we include a sample construction in Figure 1.1. Initially the cop does not know where the robber is, so he occupies the entire vertex set. We illustrate this by shading the entire graph. On her first turn we illustrate the results of her probing vertex v_1. If he answers 0, 1, 2, or 4 this would locate him, so his possible answers are 3 and 5. If he answers 3 this corresponds to response set $\{v_4, v_5\}$, and thus movement set $\{v_3, v_4, v_5, v_6\}$, as illustrated on the right. From there, a probe at v_4 locates him. If his initial response had been 5 then this would correspond to response set $\{v_7, v_8, v_9\}$, and thus movement set $\{v_6, v_7, v_8, v_9\}$. We illustrate this on the left. From there, she probes vertex v_7. His only response that does not result in being trivially captured is distance 2, corresponding to response

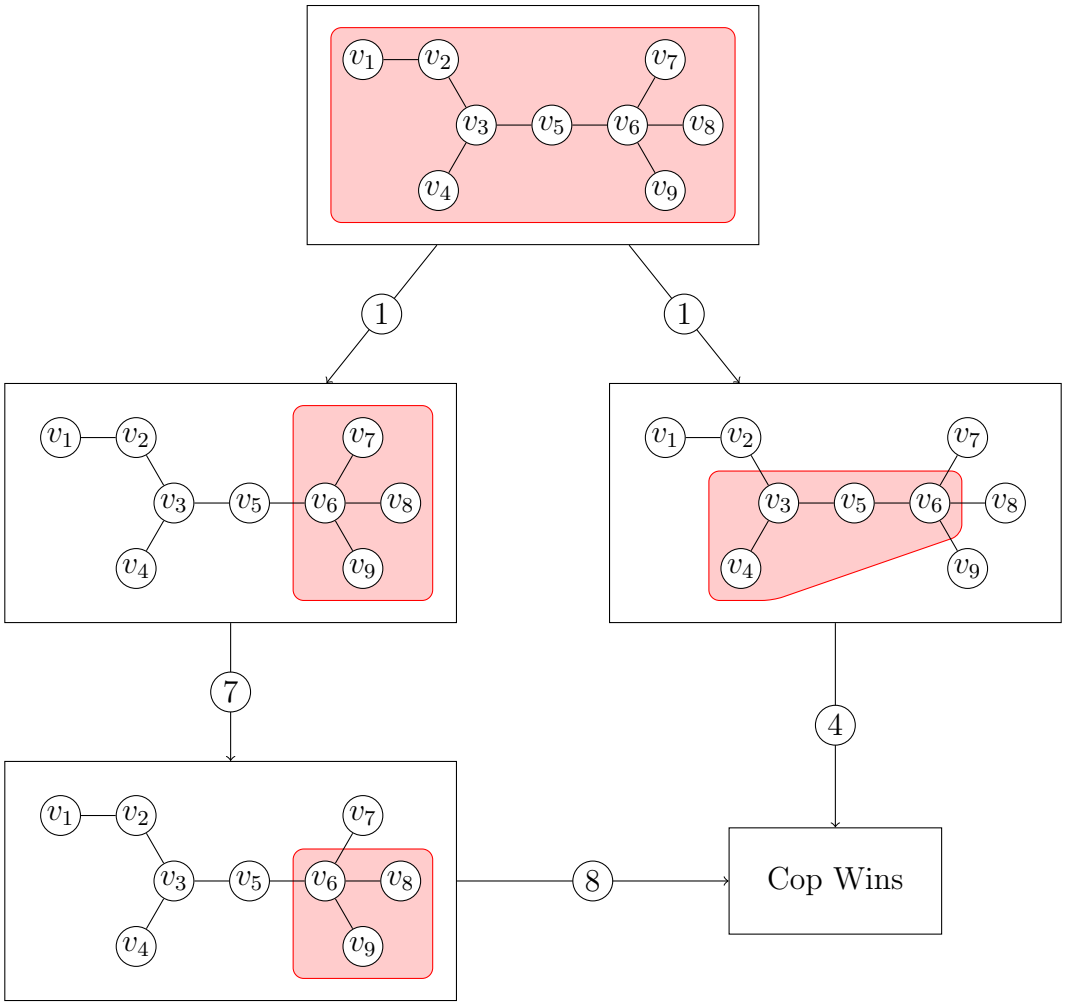


Figure 1.1: Sample moves on a pair of connected triangles

set $\{v_8, v_9\}$ and next movement set $\{v_6, v_8, v_9\}$. Then a probe at v_8 locates him.

Given the graph H we can track the game played on G in a very simple fashion by coupling the game played on G to one played on H. At time $t = 0$ the robber sits on the vertex $A_0 = V(G)$. At time t the cop chooses a vertex p_t to probe. The robber then returns d_t. In this new interpretation this is the same as the cop specifying a colour p_t of edges that she wants to restrict the robber to, and then the robber choosing which of those edges to move along. Hence any strategy for the cop will consist of specifying which vertex she would probe for each vertex set the robber might be in, and a strategy for the robber will consist of saying which edge he would take in each such case.

We now wish to establish which are the *cop-good* and *robber-good* sets, i.e. states from which the cop or robber respectively has a winning strategy. We say that a set is *0-resolvable* if it is a singleton or the empty set, and *1-resolvable* if it is not 0-resolvable and there exists a probe vertex p with the property that $|L(M, p, d)| \leq 1$ for all d. We extend this definition recursively to say that a set is *k-resolvable* if it is not l-resolvable for any $l < k$ and there exists some probe vertex p with the property that $L(M, p, d)$ is l-resolvable for some $l < k$ for all d. Hence a set is k-resolvable if there is some choice of p such that all the outneighbours of that set in colour p are resolvable in fewer than k steps. Generalising this concept we can say that a set is *resolvable* if there exists some $k \in \mathbb{N}$ such that the set is k-resolvable. We note that any resolvable set naturally gives rise to a winning strategy for the cop starting from that set, as she inductively selects the probe vertices sequentially that take the robber to sets that are resolvable in fewer queries until she forces the robber into a set that is 1-resolvable, and hence locates him. Hence any resolvable set is cop-good.

We equally want to establish what a robber-good set is in this game. In the original game a winning strategy for the robber consisted of a strategy that enabled the robber to evade capture indefinitely. As there are only a finite number of vertex sets, this is equivalent to finding a strategy that would enable him to travel round a cycle, for any possible choices made by the cop. Noting that there are only 2^n vertex sets this is the same as saying that a set M is robber-good if, for all possible choices of $p_1, \ldots, p_{2^n}$ there is a choice of distances $d_1, \ldots, d_{2^n}$ such that the sequences $R_1, \ldots, R_{2^n}$ and $M_1, \ldots, M_{2^n}$, formed by carrying out that set of probes with those answers, does not contain any response sets of size 1, and hence does not result in the robber being caught. As this forms a family of 2^{n+1} sets that the robber passes through (including his starting set), and the power set of $V(G)$ contains 2^n sets by the pigeonhole principle this family must contain a repeated set. If this is true for

all choices of $\{p_i\}$ then we conclude that the robber can specify a strategy that evades capture for any choices of $\{p_i\}$, and thus a winning strategy from the set A.

We can thus recursively build up all cop-good sets in the graph. Note that a set being k-resolvable requires it to have a $(k-1)$-resolvable neighbour (by the greedy nature of the labelling). Also note that the process by which sets are labelled cop-good is monotonically increasing, and so terminates if for any k we cannot find any $k+1$-resolvable sets. At this point the remaining sets are all labelled robber good. Thus the initial state V will either be robber-good or cop-good. This determines the locatability or otherwise of the game. □

It remains to show that this algorithm runs in finite time. Indeed, we shall give an upper bound on the time in terms of the order n of G.

Observation 2. The above algorithm runs in $O(n^3 2^n)$ steps.

Proof. Let G be a graph on n vertices with diameter D. Consider the algorithm we describe above for determining if a set is resolvable. It runs as follows:

1. Draw a graph H whose vertices are elements of the power set of $V(G)$.
2. Draw and label the edges for the graph.
3. Label the singletons and the empty set as 0-resolvable sets.
4. Recursively establish whether each set is k-resolvable for each k, starting from $k = 1$.
5. Once everything has either been labelled in step 4 or some k was found such that the graph contains no k-resolvable steps, we terminate labelling the sets as cop-good and call all remaining sets robber-good.

Stages 1, 3 and 5 take $2^n, n+1$ and 2^n steps respectively with little possible room for improvement, so we shall concentrate on stages 2 and 4.

Stage 2 requires finding $L(M, p, d)$ for all possible values of M, p and d. This can be made easier by first constructing a layered copy of G for each $v \in G$ which partitions G according to the distance from v. We call this copy G_v. If done by a breadth first search this takes at most $n + e(G) = O(n^2)$ steps for each choice of v, and hence we can complete this in $O(n^3)$ steps. We then proceed as follows. Given a vertex $M \in V(H)$ and a probe vertex $p \in V(G)$ it then takes $|M|$ steps to partition M into $\{L(M, p, d) : d \geq 0\}$, by just seeing which layer of G_p each vertex from M is in. Obtaining the $N[L(M, p, d)]$ sets from the $L(M, p, d)$ sets requires at most $2e(G)$ steps, as each edge can be used at most twice. Hence this stage takes the following number of steps, noting that $e(G) = O(n^2)$ which we need for the last line.

$$\begin{aligned}
\text{Steps taken} &\leq O(n^3) + \sum_{M \subset V(G)} \sum_{p \in V(G)} \Big(|M| + 2e(G) \Big) \\
&= O(n^3) + \sum_{M \subset V(G)} n \Big(|M| + e(G) \Big) \\
&\leq O(n^3) + \sum_{M \subset V(G)} n \Big(n + e(G) \Big) \\
&= O(n^3) + 2^n n^2 + 2^n n e(G) \\
&= O(n^3 2^n)
\end{aligned}$$

Hence Stage 2 takes at most $O(n^3 2^n)$ steps to complete.

Stage 4 would naively take in the worst case $2^{2n} \cdot n \cdot D$ steps, as for each of the possible 2^n values for k we would check each of the 2^n vertices to see if it is k-resolvable, which requires checking for each of the n possible values for p each of the D possible outneighbours in that colour. To improve this we note that it suffices to only look at the in-neighbours of sets that we have just labelled as k-resolvable when looking for $(k+1)$-resolvable sets. This will require up to $e(H)$ possible checks in total, as we shall check each vertex at most once for each edge leaving it. In order

to check a vertex efficiently the authors suggest keeping track of how many out-neighbours of each colour it has that have not yet been labelled as k-resolvable for some k. This could be established initially in $e(H)$ steps by tallying up the number of neighbours the vertex had after stage 2 in each colour and associating a n-length array with each vertex that stores this information. Then when a vertex is labelled as k-resolvable in stage 4 decreasing the array of its in-vertices by one in the appropriate colour will maintain the count of how many unresolved neighbours it has left in each colour class. Then a vertex is labelled $(k+1)$-resolvable if this process results in the count in any colour reaching zero when a neighbour is labelled k-resolvable. This process will take $2e(H)$ steps; $e(H)$ to establish the counters and $e(H)$ to then find the resolvable sets, as maintaining the arrays requires at most one step for each edge in the graph. As each vertex in H has n colours of outgoing colours, each going to at most D possible sets, $e(H) \leq 2^n \cdot n \cdot D$ and hence stage 4 will take at most $2 \cdot 2^n \cdot n \cdot D$ steps. However as $D \leqslant n$ this is at most $2 \cdot n^2 \cdot 2^n$ stage 2 remains the most time-consuming stage.

Overall this algorithm uses at most
$2^n + O(n^3 2^n) + n + 1 + 2n^2 2^n + 2^n = O(n^3 2^n)$ steps as required. □

It is also worth noting that this algorithm actually reveals an optimal strategy for the cop, by construction, as it specifies the minimal number of probes needed to locate the robber from any set $A \subset V(G)$. We now turn our attention to the complementary claim, that in the absence of a winning strategy for the cop this gives a winning strategy for the robber. This is now immediate though – since the graph H partitions the sets into robber-good and cop-good sets, the robber just picks a distance on each turn that ensures he moves to another robber-good set. Hence in the absence of a winning strategy for the cop, which corresponds to the state corresponding to V being resolvable, we get a winning robber strategy. Hence for any graph that is non-locatable there exists an explicit winning robber strategy.

A natural question to consider next is whether or not an algorithm could be found that takes fewer steps in general. We note that the longest step is stage 2, and thus any improvement to the method suggested here would give one immediate improvement. Such improvements however will not be able to bring the overall algorithm to $o(2^n)$ as it always labels every cop-good set in stage 4. This can be seen from the fact that for cop-win graphs it terminates when the initial state $V(G)$ is labelled as k-resolvable for some k, but then if $V(G)$ is k-resolvable then every subset is at most k-resolvable, and hence all of the 2^n subsets must have been examined and also labelled. In general though for cop-win graphs it is not necessary to label every subset, as it suffices to find any (or ideally a minimal) path from the initial state to singletons that the cop can force the robber down. Hence in practice there may be alternative algorithms that can take advantage of this to determine if a graph is cop-win or robber-win in shorter time. If these also did not require drawing the entire edge set of H, or could find a faster way to do so, then they could improve step 2 and thus our algorithm.

One problem with such strategies arises when considering stars however. A star has the property that a probe at the central vertex only tells the cop that the robber is not there, and prevents the robber from moving, while a probe at any leaf would also reveal if the robber was at the central vertex or not, and also eliminates that chosen leaf. Thus any strategy that only consists of probing leaves is easily shown to be strictly better than one which involves any probes at the central vertex. By symmetry all leaves are equivalent, and hence every optimal strategy is equivalent - they all just consist of probing the leaves in some order. Any such strategy will take $n-2$ turns to locate the robber, and along the way can pass through any given subset of the leaves as potential positions for the robber (by making those the last that the cop probes). Hence any set can lie on a minimal path in H from $V(G)$ to the set of 0-resolvable sets, and thus a complete algorithm will

need to examine all such sets to check for optimality. Hence we suspect that any algorithm that determined if G was cop-win or robber-win in time $o(2^n)$ will not find all optimal strategies for the underlying graph. However in general it is of more interest to determine some viable strategy in reasonable time than it is to find the provably optimal one, so any such result would still be of interest.

As this algorithm is in general time consuming we will not use it to show that any given graph is locatable or non-locatable. We shall instead give direct strategies in this dissertation. We shall however use this power-set version of the game almost exclusively, allowing the robber to occupy sets rather than individual vertices. This will simplify our descriptions of both players' strategies throughout this dissertation; hence the rest of this document can be taken as a proof of our fourth claim that this interpretation better captures how the game is analysed in practice.

CHAPTER 2

PRELIMINARY RESULTS

In this chapter we give some results about the Robber Locating Game that, although important, are sufficiently straightforward that someone new to the game would normally take them on faith. We do not consider this chapter essential reading to the rest of this dissertation, but include it purely for completeness.

We open with a simple observation that two natural assumptions about the game are justified. It seems clear that, given a graph G, if the robber has a winning strategy in which he is restricted to only moving within a certain subset of the vertices then the graph is non-locatable. We shall actually show two slightly subtly different versions of this statement. First we shall show that if he has a winning strategy that begins with him starting from a subset of the vertex set then the graph is non-locatable. Then we shall show that he can arbitrarily restrict himself to subsets, and if he can still find a winning strategy on a subset then the graph is non-locatable. Together these justify the standard intuition that if a player can win despite them being in some way restricted then they can also win with this restriction removed.

Lemma 3. Let G be a graph and $A \subset V(G)$ be a subset of the vertex set of G. Consider the restricted robber game in which he must begin in A instead of V and can only move within A, although the cop can probe anywhere in V. If S_R is a winning robber strategy for the restricted game then there exists a winning robber strategy for the unrestricted game.

Proof. We shall apply an easy coupling argument. When asked to play the unrestricted game on G the robber creates a copy of the graph, G', on which he plays according to S_R. He gives the same response distances to probes in both games. This results in him creating a set of movement sets $M'_1, M'_2, \ldots$ and response

sets $R'_1, R'_2, \ldots$ in the restricted game played on G', and movement sets $M_1, M_2, \ldots$ and response sets $R_1, R_2, \ldots$ in the unrestricted game played on G. But it is clear by construction that the sets in the restricted game are always subsets of the sets in the unrestricted game. Hence as the response sets in the restricted game always have size at least 2, so do the response sets in the unrestricted game, and thus this resulting strategy is a winning robber strategy in the unrestricted game. □

The following simple corollary gives a sufficient condition for proving that a robber strategy exists, by defining it on a suitable family of subsets. The only small technical detail, which follows immediately from the above lemma, is that when specifying robber strategies we can arbitrarily discard possible vertices that he could have moved to. This will in general simplify the descriptions of these strategies significantly. In the statement of the corollary a pair (R_i, M_i) is a valid pair of a response set and a movement set if M_i is the movement set resulting from response set R_i, i.e.

$$M_i = \{v \in V : d(v, R_i) \leqslant 1\} \tag{2.1}$$

Corollary 4. Let G be a graph, and let $R_1, \ldots, R_k$ and $M_1, \ldots, M_k$ be subsets of the vertices such that the pair (R_i, M_i) is a valid pair of a response set and a movement set for all $1 \leqslant i \leqslant k$. Further let $|R_i| \geqslant 2$ for each $1 \leqslant i \leqslant k$.

If for each $i \in [k]$ and for every $v \in V$ there exists a response that the robber can give from the set M_i that returns either a set R_j or a superset of some R_j, then the graph G is non-locatable.

Proof. The assertion follows immediately by a similar coupling argument. The robber considers a restricted game in which he is only allowed to return response sets taken from $\{R_1, \ldots, R_k\}$, and is only allowed to move to sets in $\{M_1, \ldots, M_k\}$. He couples this with an unrestricted game played on the graph G. Again it is clear

inductively that, if he gives the same distance responses in both games, then the sets he occupies in the restricted game will always be subsets of the sets he occupies in the unrestricted game. Since by assumption the response sets of the restricted game always have size at least 2, those in the unrestricted game do so as well. Hence the distance responses given by this strategy result in the cop being unable to locate the robber, and thus the graph is unlocatable. □

One may also consider a slightly weaker restricted game in which the robber is obliged to begin in a certain set $A \subset V$ but can then move freely around the graph. The same proof as above however shows that this also obeys the same property that being able to win this weaker restricted game implies that the robber can win the normal game. Hence when considering robber strategies it is sufficient to consider strategies in which he restricts himself to beginning inside, or only moving within, subsets of the vertices.

We also get the following immediate corollary about the time that the robber can survive for. Following Seager [10] we define the *location number* of a locatable graph to be the minimum number of probes needed to guarantee locating the robber, or equivalently the maximum time that a robber can survive for before being located. Extending this to all graphs we define it as infinite for unlocatable graphs, making the following corollary valid for all graphs, not just locatable ones.

Corollary 5. Let G be a graph and $A \subset V(G)$ be a subset of the vertex set of G. Then if the robber can survive for at least k probes in any of these restricted games on G, then the location number of G in the unrestricted game is at least k.

Proof. Again the robber couples a play of the unrestricted game with the restricted game. Since he can survive for at least k probes in the restricted game, there is no strategy that locates him in fewer than k probes in the restricted game. But as his movement sets and response sets in the restricted game are subsets of those in the

unrestricted game, this implies that there is no strategy that locates him in fewer than k probes in the unrestricted game as well. □

The next result we need in Chapter 5. We define a (finite) binary tree, denoted T_k, as a rooted tree with each vertex having 2 children in the layer below until you reach the leaves at distance k from the root. It is naturally partitioned into layers, with the t^{th} layer (denoted L_t) consisting of the vertices at distance t from the root – hence T_k contains $k+1$ layers, $\{L_0, L_1, \ldots, L_k\}$. The following result shows that these trees are non-locatable

Lemma 6. A binary tree with at least 5 layers is non-locatable.

Proof. We shall show this for T_4, after which the full assertion is clear due to the uniqueness of paths between vertices on trees. The proof essentially relies on defining 5 response sets and movement sets which the robber will move between. We illustrate these in Figure 2. This figure first shows T_4 with each vertex labelled clearly, to let us refer to them in this proof, and then the five subsets we shall make use of, in each case with the response set shown by a bold circle around the relevant vertices, and the corresponding movement set by background shading of the set. Thus for example $R_1 = \{v_4, v_5\}$ and $M_1 = \{v_2, v_4, v_5, v_8, v_9, v_{10}, v_{11}\}$.

We shall now outline the robber's strategy dependent on the previous movement set that he was in. We shall refer to the intuitive notation of the *parent* of a vertex v_i being the penultimate vertex on the unique path from v_1 to v_i, and a *child* of v_i as either of v_i's neighbours that is not its parent. To ease notation we shall define two sets of vertices for each choice of v, which we think of intuitively as the vertices *above* or *below* v. The former we denote by C_v^{above}, and define formally as the set of all vertices w for which the unique path from w to v passes through v's parent. For technical reasons we also include v in C_v^{above}, even though it wouldn't actually fit the above definition. The latter we denote by C_v^{below}, and comprises the compliment

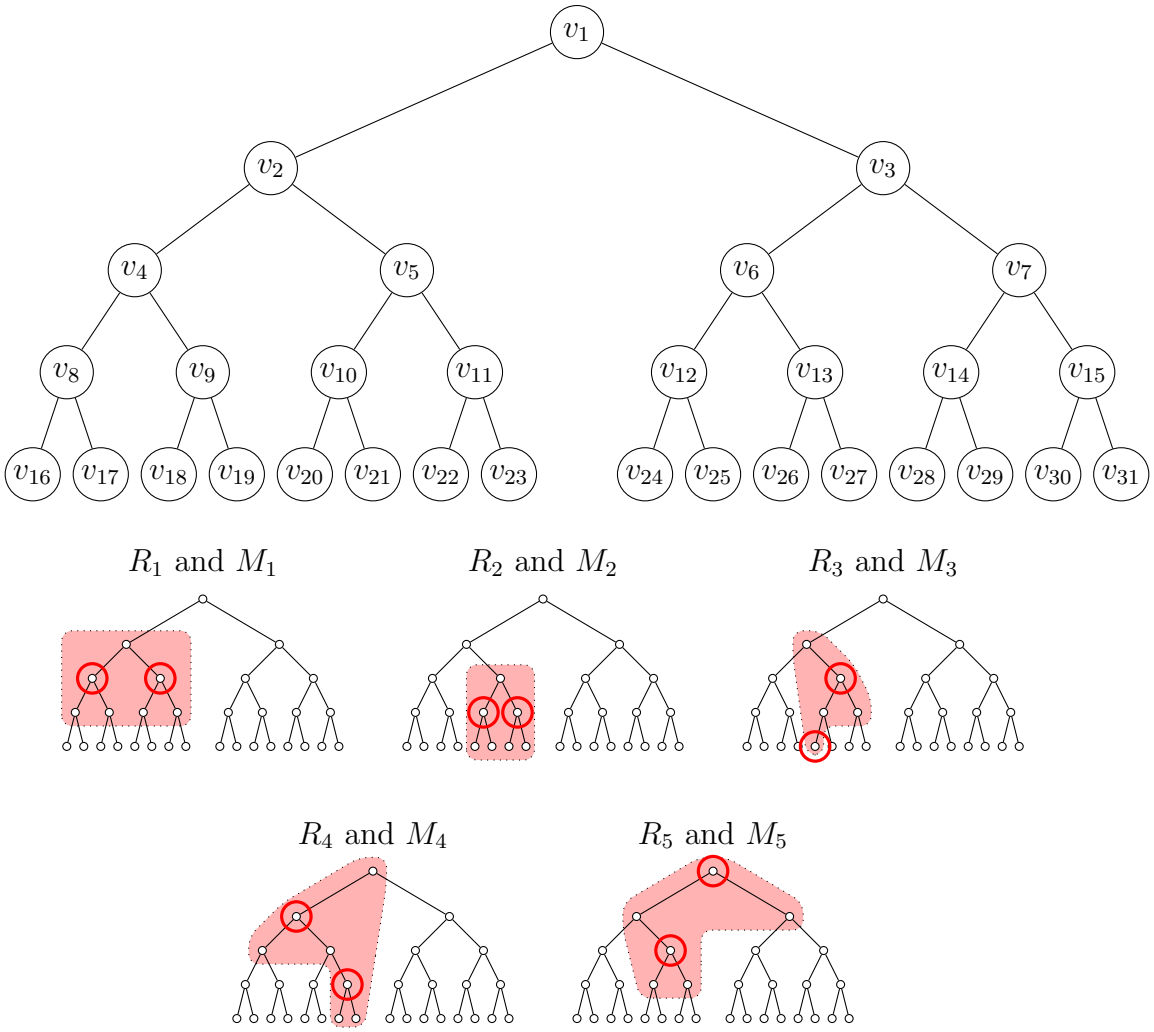


Figure 2.1: Response and movements sets for Lemma 6

of C_v^{above}, except that again we include v in C_v^{below}. Hence the set above below v contains all the vertices for which the unique path from them to v passes through one of v's children, but again with v in C_v^{below}. With these definitions in mind, we now present the robber strategy. We present a strategy on the 5 pairs of response and movement sets which suffices to show that T_4 is non-locatable by Corollary 4.

1. M_1: The cop's next probe will either be in $C_{v_2}^{\text{above}}$, $C_{v_4}^{\text{below}}$ or $C_{v_5}^{\text{below}}$. In the former two cases the cop's probe vertex will be equidistant to v_{10} and v_{11}, and the robber can return this distance, giving response set R_2 and thus moving to M_2. The final case of $C_{v_5}^{\text{below}}$ is effectively the same as $C_{v_4}^{\text{below}}$ by symmetry, so

in all three cases the robber can return something equivalent to R_2 and thus move to M_2.

2. M_2: If the cop's next probe is in $C_{v_5}^{\text{above}}$ then it is equidistant to R_2, so the robber can return this distance and thus move back to M_2. Otherwise without loss of generality by symmetry we may assume her probe is in $C_{v_{10}}^{\text{below}}$, in which case it is equidistant to a set equivalent to R_3, so the robber can return this set and move to M_3.

3. M_3: Again if the cop's probe is in $C_{v_5}^{\text{above}}$ then it is equidistant to R_2, so the robber can return this distance and thus move back to M_2. Otherwise it is $C_{v_{10}}^{\text{below}}$ or in $C_{v_{11}}^{\text{below}}$. In either case, then it is equidistant to a set equivalent to R_4 so the robber can return this set and thus move to M_4.

4. M_4: If the cop's next probe is in $C_{v_2}^{\text{above}}$, then it is equidistant to set R_1, so the robber can give this response and move to M_1. Otherwise it's in $C_{v_4}^{\text{below}}$ or $C_{v_5}^{\text{below}}$. In either case, then it is equidistant to a set equivalent to R_5 so the robber can return this set and thus move to M_5.

5. M_5. If the cop's next probe is in $C_{v_5}^{\text{above}}$ then it is equidistant to set R_2, so he can give this response and thus move back to M_2. Otherwise it's in $C_{v_{10}}^{\text{below}}$ or in $C_{v_{11}}^{\text{below}}$. In either case he can give a response that equidistant to a set equivalent to R_4, and hence move to a set equivalent to M_4.

Thus by giving appropriate responses the robber can cycle between these 5 sets, or sets equivalent to them, for any choices of probes by the cop, and hence avoid location indefinitely. □

The next lemma we require for our chapter on infinite graphs. Essentially it shows that attaching too simple a pendant structure to a non-locatable graph cannot help the cop.

Lemma 7. Let H be a non-locatable graph, and let G contain an induced subgraph H' which is isomorphic to H with only one edge connecting H' to $G \setminus H'$ in G. Then G is non-locatable.

Proof. The result follows almost immediately from Lemma 3. The robber again couples the game on G with a copied game on H, and commits to being restricted to H' in G. Note that by Lemma 3 a winning strategy under this restriction implies that G is non-locatable. Let v be the vertex in H' that connects to the rest of the graph G. On each turn the cop either probes a vertex in H' or in the complement of H' in G. If the cop probes a vertex in H' then the robber considers the cop to have probed the corresponding vertex in H, and plays according to his winning strategy in H, returning the prescribed distance to the cop. If the cop probes a vertex in G then the robber imagines that the cop probed v, and returns whatever distance would be prescribed by his winning strategy in H plus the distance from v to the probed vertex in G. This ensures that the movement and response sets in H' that he occupies will be the same as those he occupies in the game on H, and since his strategy is a winning strategy on H it is thus also a winning strategy on H', and thus G is non-locatable. □

CHAPTER 3

SUBDIVISIONS OF FINITE GRAPHS

The work in this chapter is all joint work with Sebastian Koch of Cambridge and John Haslegrave of Sheffield. Following Carragher, Choi, Delcourt, Erickson and West [3], given a graph G and a positive integer m, we write $G^{1/m}$ for the graph obtained by replacing each edge of G by a path of length m through new vertices. This notation is chosen to compare with graph powers, in which G^k is the graph formed from G by adding edges between all pairs of vertices whose distance in G is at most k. Thus although $\left(G^{1/m}\right)^m \neq G$, and $\left(G^m\right)^{1/m} \neq G$, $\left(G^{1/m}\right)^m|_G$, the graph induced by the original vertex set of G in $\left(G^{1/m}\right)^m$, is equal to G. Each such path is called a *thread*, and an *original vertex* in $G^{1/m}$ is a vertex which corresponds to a vertex of G. We denote the thread linking two vertices v_i and v_j by $v_i \cdots v_j$. The *span* of an original vertex consists of all the vertices at distance less than m from it, which includes the vertices along the threads leaving that vertex, but not the far endpoints of those threads.

In [3] the authors show that subdividing any finite graph sufficiently many times yields a locatable graph, and show that it is always sufficient to subdivide each edge by a factor of $\min\{n-1, \max\{\mu(G) + 2^{\mu(G)}, \Delta(G)\}$, where $\mu(G)$ is the *metric dimension* of G. The notion of metric dimension was introduced independently by Slater [12], and by Harary and Melter [7]. The metric dimension of G is the size of the smallest set S of vertices such that for every $x, y \in V(G)$ with $x \neq y$ there is some $z \in S$ with $d(x, z) \neq d(y, z)$. We shall however not focus on this concept, we concentrate on the other half of the inequality, the upper bound of $n-1$. As well as this general bound (which they conjecture to be necessary for a complete graph) they show that for a complete bipartite graph $K_{a,b}$, $m \geqslant \max\{a, b\}$ is sufficient for $K_{a,b}^{1/m}$ to be locatable, and conjecture that this is also necessary.

Our first main results show that both of these conjectures are false. In fact we

shall establish the exact thresholds for m which make $K_n^{1/m}$ and $K_{a,b}^{1/m}$ locatable. This is slightly complicated in the former case since although the threshold is generally $n/2$, there are some small counterexamples. As such we state our theorem for $n \geq 14$ to avoid these, and shall address them specifically when we come to present the proof.

Theorem 8. Let $n \geqslant 14$. Then $K_n^{1/m}$ is locatable if and only if $m \geqslant n/2$.

For the complete bipartite case, the bounds of [3] can be improved to be phrased in terms of the smaller of the bipartite graph's vertex classes

Theorem 9. The graph $K_{a,b}^{1/m}$ is locatable if and only if

$$m \geqslant \begin{cases} \min\{a,b\} - 1 & \text{if} \quad \min\{a,b\} \geqslant 4 \\ \min\{a,b\} & \text{if} \quad \min\{a,b\} \leqslant 3 \end{cases} \tag{3.1}$$

Further we can extend the method used in the above proofs to show an improved lower bound on sufficient subdivision to make a graph locatable, which is asymptotically the same as these and hence essentially optimal

Theorem 10. Let G be any graph on n vertices. Then $G^{1/m}$ is locatable if $m \geqslant n/2 + 1$.

3.1 Complete graphs

We shall present the proof for Theorem 8 in full detail, and then refer to it in the proofs of the other two results as they are very similar. To show Theorem 8 we shall present the two halves of the statement separately. We begin with the proof that the graph is non-locatable for $m < n/2$. This will rely on the robber being able to move between original vertices without being located by the cop.

Theorem 11. Let $m < n/2$. Then $K_n^{1/m}$ is non-locatable.

Proof. We prove this by giving an explicit strategy for the robber that achieves the following: assuming at some time he could be in a set of two original vertices, then we claim he can either remain in this pair of original vertices, or reach another pair without being located, and hence he can evade capture indefinitely. We denote the set of original vertices $\{v_1, \ldots, v_n\}$.

Let us first assume that following a probe by the cop (which we shall refer to as the 0^{th} probe) the robber reveals that he could be in the pair of original vertices $\{v_1, v_2\}$ (relabelling if necessary), but that the cop does not know his location. After this he can move to anywhere in $N[v_1] \cup N[v_2]$. Firstly we shall separately consider the result of the cop's first probe, which can be in one of two places.

1. If the cop's probe was equidistant to v_1 and v_2 then the robber can claim to have remained in $\{v_1, v_2\}$, and thus still be in $N[v_1] \cup N[v_2]$ after the probe. If the cop always probes vertices that are equidistant from v_1 and v_2 then the robber can repeat this, evading capture indefinitely.

2. If the cop's probe was not equidistant to v_1 and v_2 then it was on a thread incident to at least one of them. Let us call the vertex she probes here p. Without loss of generality we may assume both that this probe is her first probe (ignoring any that were equidistant to $\{v_1, v_2\}$ and came before it), and that it is in the span of v_1. Following this probe the robber will now adopt his motivating strategy of moving towards a new original vertex. He can thus return the distance $d(v_1, p) + 1$, claiming that he was at v_1, and so moved to the neighbourhood of v_1 at the previous step. He will then continue moving down some thread towards another original vertex. Given that the robber now commits to follow this strategy the cop only needs to determine his destination before he reaches it. We shall show that this is not possible by

keeping a count of how many threads the cop has not yet eliminated. This first probe only eliminates the thread that p is on, so following it the cop knows that the robber was at distance 1 (and is now at distance 2) from v_1 and is moving along one of $n-2$ possible threads.

Each subsequent probe can eliminate at most 2 threads for the robber, since probing anywhere inside a thread from v_1 eliminates only that thread and probing inside $v_i \cdots v_j$ eliminates only $v_1 \cdots v_i$ and $v_1 \cdots v_j$. The robber can then remove those from his possible destinations and continue moving away from v_1. Hence after t steps the robber is at distance $(t+1)$ from v_1 and at most $2t-1$ threads have been eliminated. After the $(m-1)^{\text{st}}$ step the robber reaches the remaining possible original vertices that he could have been heading towards. There were initially $n-1$ threads that he could have been heading down, and so after $(m-1)$ steps he could be inside any of at least $n-1-(2(m-1)-1) = n-2m+2 \geqslant 3$ threads leaving v_1.

There are two possible scenarios to consider. Firstly, if as described above, the cop eliminates 2 threads on every probe except the first, then she would be unable to determine if the robber had gone halfway down a thread (pausing at the first near midpoint for a step if m is odd) and then returned to v_1. Hence in this case after the $(m-1)^{\text{st}}$ probe the robber could move to any of at least 4 original vertices (those at either end of the uneliminated threads). If the cop did check to see if the robber turned around she would have to do so by probing on a vertex on a thread of v_1, and this would only eliminate one thread on that turn. This would mean she would eliminate one fewer thread, leaving him at least 4 threads he could be inside after $(m-1)$ steps and thus at least 4 original vertices he could reach. In either case he can move into a set of at least 4 original vertices. The next probe by the cop must lie on some thread between at most 2 of them, so at least 2 will be equidistant to the next probe. The robber can now claim to have moved into that pair, and so can reach another pair of original vertices as required. Repeating this process lets

him avoid capture indefinitely. □

We now turn our attention to the bound for a graph to be locatable. We shall show that if $m \geqslant n/2 + 1$ then the cop can follow a simple strategy to locate the robber, which proceeds in three stages. This argument can be improved in the third stage to also cover the situation where $m = \lceil n/2 \rceil$ provided that $m \geqslant 7$ (i.e. $n \geqslant 14$), so we include both of these conditions in the following theorem.

Theorem 12. The subdivided graph $K_n^{1/m}$ is locatable if either of the following conditions holds

1. $m \geqslant n/2 + 1$
2. $m \in \{n/2, (n+1)/2\}$ and $m \geqslant 7$

Proof. Our strategy for the cop runs in three stages. In the first stage she forces the robber to move to an original vertex, although she does not attempt to control which. In the second stage she narrows down the set of original vertices that he could be in to a set of size 2. In the final stage she locates him. Either of the two conditions for the theorem will suffice for Stages 1 and 2, in Stage 3 her strategy will vary slightly according to which condition is satisfied.

In Stage 1 the cop probes all the original vertices in any order until she either gets an answer equal to m or finds two original vertices at distance less than m from the robber. If she gets an answer equal to m then she knows he has entered an original vertex, and moves to Stage 2. If this does not happen then he must have remained inside a single thread. When probing either end of it she would get an answer less than m, and by noting which two original vertices this occurs on she can identify which thread he is inside, and locate him. Thus either the robber is located or the cop moves to Stage 2.

In Stage 2 the cop wishes to narrow down the set of possible original vertices the robber could be in to a set of size 2. She will do this by eliminating candidates,

so let us now re-order the original vertices as $v_1, \ldots, v_n$ such that v_1 is the last original vertex that she probed in Stage 1 – hence the robber is known not to be at v_1 at the start of Stage 2. Throughout she will track the candidates she has eliminated by maintaining a counter r which is the index of the last vertex that she eliminated. Hence we set $r = 1$ initially, and throughout this stage having eliminated the vertices up to v_r she will be trying to eliminate v_{r+1} and thus increment r. We can assume throughout that $r < (n-2)$, as once she has eliminated v_{n-2} there are only two vertices left, and she can proceed to Stage 3.

To eliminate v_{r+1} the cop begins by probing this vertex, which can give one of five possible responses. Three of these cases are simple to deal with:

1. *The distance is* 0. The cop has found the robber and wins the game.

2. *The distance is* $m-1$. The cop then knows that the robber was at an original vertex of higher index, and that he has left it, moving towards v_{r+1}. The cop can now force the robber to return to the original vertex that he came from by alternately probing v_{r+1} and the remaining original vertices with indices higher than $r+1$ in order. If the robber moves to v_{r+1} the cop will detect this and thus locate him easily, and if he does not return then she will eventually find the vertex he came from, and thus locate him. Hence he must return, which she will detect when she gets distance m. Along this process she will potentially eliminate not just v_{r+1} but possibly many more candidates – she proceeds by setting r to the highest index that she has eliminated, and probing the next original vertex.

3. *The distance is* m. The cop concludes that the robber is still at an original vertex of higher index than $(r+1)$. She increases r by 1, and repeats the process by probing the next original vertex.

4. *The distance is* $m+1$. This is the most complicated case to deal with. The

cop now concludes that the robber was at an original vertex of higher index, say v_i, and has left it moving towards another original vertex, say v_j. She now has two situations to consider. If $j \leqslant r$ then identifying v_j before he reaches it will let her force him back into v_i as in case (ii) above. If $j > r$ (and thus $j > r+1$ as if $j = r+1$ then the distance would have been $m-1$ which was case (ii) above), then she is less concerned with finding v_j, it suffices for her to force him to either v_i or v_j, as then she can continue with the above process having eliminated all the original vertices up to v_{r+1} as required. She will therefore address these situations sequentially.

Firstly the cop establishes whether $j > r$ by checking all the vertices in $v_1, \ldots, v_r$ to see if they are the destination for the robber. Ideally on each turn she would check two possible destinations by probing the midpoints of the threads linking the first r original vertices. In general she will not be able to do this for the first step, but she can begin by probing v_1 which eliminates that as a destination.

(a) If the robber announces distance m then he has returned to v_i, and the cop can continue Stage 2 with v_{r+1} eliminated.

(b) If the robber gives distance $m+1$ then he is still at distance 1 from v_i, and the cop can continue to probe through the set $\{v_2, \ldots, v_r\}$ until he moves in either direction or she eliminates all of them – in the latter case we move to the next paragraph which outlines what to do once they have all been eliminated.

(c) If at some point the robber answers $m+2$ then the cop knows he was not heading to the vertex just probed but has moved to the second layer of vertices from v_i. From this point she can eliminate two vertices from $\{v_2, \ldots, v_r\}$ at each step by either probing midpoints if m is even or

near-midpoints if m is odd. Either way she can tell whether he moves back towards v_i, in which case she moves back to probing single vertices once he gets back to the first layer to identify the exact moment he returns to v_i, or keeps eliminating pairs if he does not. If he continues to head away from v_i, then by eliminating two vertices at each step she can eliminate $2(m-3)+1$ before he reaches another original vertex. But as $m > (n-1)/2$ and there were only at most $n-3$ original vertices in $\{v_1, \ldots, v_r\}$ this leaves at most two such vertices that he can reach. If there is only one vertex then by probing it directly she can determine if he has reached it (in which case she locates him), or if he remains inside the thread between it and v_i, in which case she can force him back to v_i by alternately probing v_j and the remaining original vertices. If he could have reached a pair, by probing a vertex on the thread between these last two vertices she can distinguish if he is in this pair, allowing her to move to Stage 3 if he is. Hence if he tries to move towards $\{v_1, \ldots, v_r\}$ she will either locate him, move to Stage 3 or force him back into v_i.

Thus having established that the robber was not moving towards $\{v_1, \ldots, v_r\}$ it remains for her to deal with the situation where $j > r$. If he left the $v_i \cdots v_j$ thread either by reaching v_i during the above probes or v_j on the last probe, the cop will detect this when v_i or v_j is reached, allowing her to repeat Stage 2 having eliminated v_{r+1}. If he has not left this thread she can then probe the original vertices with indices higher than $r+1$ to eliminate those directly until she finds either of v_i or v_j – in which case she would proceed as in Case (ii) to force him into the other end of the thread, and repeat Stage 2 with more vertices eliminated.

5. *The distance is 1.* This is the final case to consider. In this case the cop has found the original vertex that the robber was at, and he has moved 1 away from it. The strategy here is very similar to case 4: she first makes sure that he is not moving towards $\{v_1, \ldots, v_r\}$, making sure to note if he returns to v_i, and then afterwards continues checking the remaining vertices in pairs. Carrying out the same analysis shows that in most cases he will be located when he returns to v_i, the only case when he is not is when he either moves halfway down a thread towards $\{v_{n-2}, v_{n-1}, v_n\}$ and then moves back to v_i or goes all the way to v_n. But as this is the only case where the cop does not locate him directly if it occurs she will know, and thus be able to move to Stage 3 knowing he is at either of v_i or v_n.

We now move to Stage 3, which starts after the cop makes some probe and knows the robber is at one of two original vertices. We shall label them as $\{a, b\}$, and note that he can move to the neighbourhood of them before the cop's first probe in Stage 3. For her first probe the cop probes the vertex at distance 1 from a on the $a \cdots b$ thread. This allows her to distinguish whether the robber was at a or b before, and whether he is inside the $a \cdots b$ thread now or another one. The cop wins immediately unless the robber answers distance 2 or distance m, in which case he has left the initial vertex he was at and moved towards an initial vertex other than a or b. Without loss of generality we shall assume he was at a and is thus now moving to one of the other $(n-2)$ possible locations, noting that this first probe reveals him to be at distance 1 from a.

The cop's strategy now reduces to finding which thread the robber is inside before he can reach the other end of it, being sure to note if he returns to a. The second probe varies according to whether m is even or odd. If m is even then the cop probes a midpoint of a thread between two original vertices that have not been eliminated yet, whereas if it is odd then the cop probes a neighbour of a

near-midpoint, say the vertex on the $c \cdots d$ thread that is distance 3 further from d than c. In either case we can distinguish whether the robber is heading to that pair or not – the one slightly complicated case is if m is odd and he responds with $(m-1)/2+m$ in which case he could have remained at distance 1 from a or be distance 2 from a heading specifically towards d. If following this probe the robber uses the fact that he could have been distance 2 from a to move to the vertex distance 3 from a the cop will notice on her next probe and locate him. In this case the cop can therefore assume that the robber is at distance 1 and effectively eliminate c from the possible destinations, doing so without him having moved closer to another original vertex so effectively for free. Thus the cop's second probe can always eliminate two possible destinations for the robber – and by probing at midpoints if m is even or near-midpoints if m is odd this also holds for the subsequent probes.

After t probes the robber will be within distance t of a, and the cop will have eliminated $1+2(t-1)$ possible destinations. If condition 1 was satisfied (i.e. $m \geqslant n/2+1$) then after $(m-1)$ probes there are no possible destinations left – hence he cannot reach another original vertex and so would be located. If alternately condition 2 was satisfied (i.e. $m \in \{n/2, (n+1)/2\}$ and $m \geqslant 7$ then after $m-1$ probes there are only at most three original vertices left that he could be moving between, two possible destinations (which we shall refer to as v and w) and a. Including the possibility that he turned around at the midpoint or near-midpoint (according to the parity of m), and assuming at each step he continued to move (as otherwise it is easier to locate him), this means that following the $(m-1)^{\text{st}}$ probe he is either distance $0, 1, 2, (m-2), (m-1)$ or m from a inside either the $a \cdots v$ thread or the $a \cdots w$ thread. However, in this case he can be located by probing the vertex at distance 2 from v along the $a \cdots v$ thread, provided $m \geqslant 7$, so hence he can be located even in this worst case scenario, completing the proof. □

This answers the question of when $K_n^{1/m}$ is locatable or non-locatable for all but a small number of cases, namely $K_3^{1/2}$, $K_4^{1/2}$, $K_5^{1/3}$, $K_6^{1/3}$, $K_7^{1/4}$, $K_8^{1/4}$, $K_9^{1/5}$, $K_{10}^{1/5}$, $K_{11}^{1/6}$ and $K_{12}^{1/6}$. These cases can be checked directly – note that $K_3^{1/2} = C_6$ which Carraher, Choi, Delcourt, Erickson and West showed in [3] is non-locatable. We omit the details of the case checking, which was carried out exhaustively, and summarise them in the table below. An 'L' indicates that the graph in question is locatable, and an 'N' that it is non-locatable. To aid legibility we have shaded the non-locatable cases, and included a line alone the $n/2$ bound for m. This highlights that the general rule is the $K_n^{1/m}$ is locatable if and only if $m \geqslant n/2$, aside from a few counterexamples.

		Number of vertices, n																			
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Subdivision amount, m	1	L	L	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N
	2	L	L	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N
	3	L	L	L	L	L	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N
	4	L	L	L	L	L	L	L	L	N	N	N	N	N	N	N	N	N	N	N	N
	5	L	L	L	L	L	L	L	L	L	N	N	N	N	N	N	N	N	N	N	N
	6	L	L	L	L	L	L	L	L	L	L	L	L	N	N	N	N	N	N	N	N
	7	L	L	L	L	L	L	L	L	L	L	L	L	L	L	N	N	N	N	N	N
	8	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	N	N	N	N
	9	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	N	N
	10	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L

3.2 Bipartite graphs

We now turn our attention to complete bipartite graphs. In this setting we are able to determine the winning player on $K_{a,b}^{1/m}$ for any a, b, and m. Throughout this section we shall write A and B for the sets of original vertices in $K_{a,b}^{1/m}$ corresponding to the two vertex classes of $K_{a,b}$, with $|A| = a$ and $|B| = b$. Recall that we are trying to show an improved form of the result of [3], namely that $K_{a,b}^{1/m}$ is locatable if and only if

$$m \geqslant \begin{cases} \min\{a,b\} - 1 & \text{if} \quad \min\{a,b\} \geqslant 4 \\ \min\{a,b\} & \text{if} \quad \min\{a,b\} \leqslant 3 \end{cases} \tag{3.2}$$

We shall prove this through several lemmas for the various parts. First, we show the claims on non-locatable graphs. Observe that if $m = 1$ and $\min\{a,b\} \geqslant 2$ then the graph is non-locatable, as for each probe the robber can claim to be in the other part of the graph, and hence can never be located. We first show that $K^{1/m}_{a,b}$ is non-locatable if $\min\{a,b\} \geqslant 4$ and $m \leqslant \min\{a,b\} - 2$ in Lemma 13. This leaves one final case, $K^{1/2}_{3,b}$ for $b \geqslant 3$, which we shall cover in Lemma 14.

Lemma 13. If $\min\{a,b\} \geqslant 4$ and $m \leqslant \min\{a,b\} - 2$ then $K^{1/m}_{a,b}$ is non-locatable.

Proof. We shall prove the stronger statement that the robber wins even if he is required to be at an original vertex for every m^{th} probe, alternating between A and B, so that he is in A at the time of the km^{th} probe for every even k. We show that, provided the cop has not won after the km^{th} probe, the robber can ensure that she has not won by the $(k+1)m^{\text{th}}$ probe. For ease of writing, we assume that k is even.

Suppose that the robber is at $u \in A$ for the km^{th} probe, but that the km^{th} probe does not locate him uniquely. We show that, no matter which vertices the cop probes, there are two possible threads for the robber to travel along between the km^{th} and $(k+1)m^{\text{th}}$ probes, which the cop is unable to distinguish between, so that she will not be able to win by time $(k+1)m$. Suppose her $(km+l)^{\text{th}}$ probe (for some $1 \leqslant l \leqslant m$) is at vertex z, which is inside the thread $x \cdots y$ for some $x \in A$ and $y \in B$. For each $v \in B$, write $w_{v,l}$ for the vertex on the thread $u \cdots v$ at distance l from u. If $x \neq u$ then for any $v \neq y$ we have

$$d(z, w_{v,l}) = \min\{d(z,x) + 2m - l, d(z,y) + m + l\}$$

Equally if $x = u$ then, again for any $v \neq y$, we have $d(z, w_{v,l}) = d(z,x) + l$.

Suppose that for each l with $1 \leqslant l \leqslant m$ the answer consistent with the robber being at any one of the vertices $w_{v,l}$ for $v \neq y$ is received from the $(km+l)^{\text{th}}$ probe. Then each probe eliminates at most one of the threads leaving u, and since m probes have been made, and $m \leqslant \min\{a, b\} - 2$, at least 2 remain, so the cop has not yet won. □

Next we complete the classification of the non-locatable subdivided complete bipartite graphs by showing that for any $b \geqslant 3$, $K_{3,b}^{1/2}$ is non-locatable. This is the remaining non-locatable graph from Theorem 9

Lemma 14. If $b \geqslant 3$, $K_{3,b}^{1/2}$ is not locatable

Proof. Suppose that after the cop's t^{th} probe there are two possible locations for the robber which are both in A or both in B, say u and v with $u, v \in A$. We show that the robber can ensure that there are still two possible locations, both in A or both in B, either after the $(t+1)^{\text{st}}$ probe or after the $(t+2)^{\text{nd}}$. If the $(t+1)^{\text{st}}$ probe is equidistant from u and v this is trivial, as the robber can return the distance to u or v. If the $(t+1)^{\text{st}}$ probe is u or v then all neighbours of u will be equidistant, so the robber can claim to be at one of them. Any vertex in A is equidistant from all vertices in B, and any other vertex is equidistant from all but one of the vertices in B, so no matter what vertex the cop chooses for her $(t+2)^{\text{nd}}$ probe, there will be at least $b - 1 \geqslant 2$ vertices in B at the same distance from it. By this point the robber can have reached any of these without being caught. The only remaining case is for the $(t+1)^{\text{st}}$ probe to be at a vertex which is adjacent to either u or v, say the vertex w between u and x with $x \in B$. Let y and z be two other vertices in B. The midpoints of the threads $u \cdots y$, $u \cdots z$ and $v \cdots x$ are all at distance 2 from w, so if the robber has moved to one of these the cop cannot determine which. Then no matter which vertex the cop probes at time $t+2$, some two of x, y and z are at the same distance, and so the robber can ensure there are two possible locations in B

after this probe. □

This concludes the proofs for the non-locatable half of the assertion of Theorem 9. Next we show that in all other cases $K_{a,b}^{1/m}$ is locatable. Note that the cop can win on the star $K_{1,b}$ by probing leaves in turn, and in general probing the vertices of degree 1 will also locate the robber on $K_{1,b}^{1/m}$ for any $b \geqslant 1$ and any $m \geqslant 1$. Hence $K_{1,b}^{1/m}$ is locatable for any $b \geqslant 1$ and any $m \geqslant 1$. This covers the case $\min\{a, b\} = 1$. Next we deal with the case $\min\{a, b\} = 2$.

Lemma 15. The graph $K_{2,b}^{1/m}$ is locatable for any $b \geqslant 2$ and for any $m \geqslant 2$.

Proof. We begin with the case $m = 2$. Write x and y for the two vertices in A. Let the cop start by probing x. If she receives the answer 2 the robber is in B. If the answer is 0 or 4 she has won. If it is 1 or 3 she knows that the robber is adjacent to x or y respectively.

Now we show that the cop can win from a position in which she knows that the robber is in a particular subset of the neighbourhood of x (or, equivalently, if she knows the robber is in a particular subset of the neighbourhood of y), and she can win from a position in which she knows that the robber is in a fixed subset of B. We prove both simultaneously by induction on the size of the subset, k. In each case if $k = 1$ she has already won.

If the robber was just at one of k neighbours of x, the cop probes one of the k adjacent vertices of B. If the answer is at most 2 then the robber is caught. If the answer is 3 then he is known to be at one of $k - 1$ neighbours of x and if it is 4 he is known to be at one of $k - 1$ vertices of B; in either case we are done by induction.

If the robber was just at one of k vertices in B, the cop probes one of these. An answer of 2 is impossible, and if the answer is 1 then she can win by next probing x. If the answer is 4 then she knows the robber is at one of $k - 1$ vertices of B, and we are done by induction. If the answer is 3 then she probes x next; now if the answer

is 0 or 4 she has won, and if it is 1, 2, or 3 she has reduced to one of $k-1$ vertices adjacent to x, in B, or adjacent to y respectively, so we are done by induction for the case $m = 2$.

If $m > 2$ the same argument works, except that the cop differentiates between answers that are less than or greater than m at the first step, to determine if the robber is on a thread from x or y. The remaining proof continues in the same manner, with the cop just tracking which threads the robber could be on, until the cop knows which thread he is on or locates him passing through an endpoint. Assuming the former, when $m = 2$ she would be done here, as threads only have 1 interior vertex, if $m > 2$ she requires one additional probe at either endpoint of it to locate him. □

Finally we deal with the case $\min\{a, b\} \geqslant 3$. Again we present this is a slightly different way from that in Theorem 9 – note that the following statement covers both $K_{3,b}^{1/m}$ for $m \geqslant 3$ and $K_{a,b}^{1/m}$ for $\min\{a, b\} \geqslant 4$ and $m \geqslant \min\{a, b\} - 1$.

Lemma 16. Let $\min\{a, b\} \geqslant 3$. If $m \geqslant \min\{a, b\} - 1$ and $m \geqslant 3$ then $K_{a,b}^{1/m}$ is locatable.

Proof. Suppose $a \leqslant b$. Again we give a two-stage winning cop strategy. In the first stage we show that the cop can win or establish that the robber is in B, and in the second stage we show that she can win once she knows that the robber is in B.

In the first stage, the cop probes vertices in A in turn until she receives an answer of m (indicating that the robber is in B) or less than m. This must eventually happen, since if the robber does not reach B he must remain nearer one particular vertex in A than any other, and when the cop probes this vertex she will get an answer of less than m. In this case write x for the vertex in question. Once the cop has found x, the robber cannot leave his current thread without moving either to x or to some vertex in B, so the cop then probes vertices in B until she

receives an answer of $2m$ (indicating that the robber is in B) or at most m (in which case she can determine his location).

In the second stage we show that the cop may win from a position where the robber is known to be in a fixed subset of B, by induction on the size of the subset, k. This is true for $k = 1$ as she has already won. If $k > 1$ then write B' for the set of k vertices in question. The cop starts by probing the vertex adjacent to B' inside the thread $x \cdots y$ for some $x \in A$ and $y \in B'$. The possible answers are

1. 0 if the robber is at the probed vertex,
2. 1 if he is at y,
3. 2 if he is at some other neighbour of y,
4. $2m - 2$ if he is on another thread leading to x,
5. $2m - 1$ if he is at a vertex of B' other than y,
6. $2m$ if he is inside a thread which does not include x or y.

Since $m \geqslant 3$, these cases are all different. An answer of 0 or 1 is an immediate win for the cop, and after an answer of $2m - 1$ she wins by the induction hypothesis. After an answer of $2m - 2$ the cop probes vertices of B' until either she receives an answer of at most m, winning, or she receives an answer of $2m$, in which case she knows the robber is at one of at most $k - 2$ vertices of B' and she wins by the induction hypothesis.

After an answer of 2 or $2m$, the robber must be inside a thread which does not reach x. The cop now probes vertices of A, other than x, in turn, until she receives an answer of $2m$, m, or less than m. One of these must eventually happen since either the robber reaches one end of the thread he is currently inside, or he remains in the same thread until such time as the cop probes its end in A.

1. If the answer $2m$ occurs first, the cop knows that the robber has reached some vertex $u \in A$ which is neither x nor one she has probed since the robber left B. Since the robber has taken at least m steps to reach A, she has probed at least $m-1$ vertices in A, and together with x she has eliminated at least $m \geqslant a-1$ vertices of A, so there is only one possibility and the robber is caught.

2. If the answer m occurs first then the robber is at a vertex of B', and, since the cop knows whether or not this is y, she has either caught the robber or reduced to a set of $k-1$ vertices, so wins by the induction hypothesis.

3. If an answer less than m occurs first, say when probing u, then the cop has won if that answer is 0, or if the robber was initially known to be inside a thread meeting y. Otherwise, she knows that the robber is inside some thread $u \cdots v$ for $v \in B' \setminus \{y\}$; now she proceeds by probing vertices of $B' \setminus \{y\}$ in turn until she receives an answer of at most m (in which case she has won) or of $2m$ (in which case she knows that the robber is at some vertex in $B' \setminus \{y\}$, and so wins by induction).

□

This concludes our classification of subdivided complete bipartite graphs. If $\min\{a,b\} \geqslant 4$ then $K_{a,b}^{1/m}$ is locatable if and only if $m \geqslant \min\{a,b\} - 1$, whereas if $\min\{a,b\} \leqslant 3$ then $K_{a,b}^{1/m}$ is locatable if and only if $m \geqslant \min\{a,b\}$.

3.3 General graphs

We now turn our attention to the advertised main result, that any graph G is made locatable by subdividing the edges by a factor of $n/2+1$. We do not claim that this is always necessary – for example G may already be locatable without any subdivision – but it does form a tight upper bound since, for example, K_4 does

require subdivision by a factor of at least 3 to be locatable. This therefore constitutes an optimal generally sufficient subdivision condition. Note that, as we are only showing that this level of subdivision is sufficient, we only need to give a cop strategy that works on $G^{1/m}$ for all $m \geqslant n/2 + 1$.

Proof. Let I be a largest independent set in G, let M be the vertex set of a largest matching in $V \setminus I$, and let X be the remaining vertices, such that V is partitioned into I, M and X. With slight abuse of notation, let I, M and X also refer to the original vertices in $G^{1/m}$ corresponding to I, M and X in G; since we shall work exclusively in $G^{1/m}$ this distinction should not be confusing. We give a cop strategy that achieves the following:

1. forces the robber to enter an original vertex and reveals when he does so;
2. forces the robber to enter I and reveals when he does so;
3. locates the robber once he is known to be in I before he can pass through another original vertex.

We shall present a strategy that works for $m \geqslant |X| + \frac{|M|}{2} + 1$, which it is simple to show is at most equal to $n/2+1$ since $|X| \leqslant |I|$ as X must also be an independent set. We shall assume that m is even, and hence the edges in the matching behind M have midpoints, for clarity – if not, replace the word 'midpoint' with 'near midpoint' in the proof below and make the same adjustments as in the proof of Theorem 12.

First, the cop can force the robber to enter some original vertex by probing all of them in turn. If he does enter an original vertex he will return a distance that is a multiple of m, and thus the cop will know he has done so, allowing her to move to Step 2. If he does not then he remains on some $v_i \cdots v_j$ thread, and since when she probes either v_i or v_j she will receive a response less than m she will determine both v_i and v_j, and hence locate him. Thus to avoid being detected in Step 1 he must move to an original vertex, allowing the cop to move to Step 2.

At the beginning of Step 2 the cop knows that the robber was in an original vertex, which may or may not have been in I. We give a strategy for her which would locate him if he had started in I, and if not forces him to enter I. This strategy is very simple – she probes the midpoint of each edge in the matching that gave rise to M, and then each vertex in X. We claim this works by separately considering what he may have done during this process – note that he can only reach another original vertex at the earliest following the final probe by the choice of m, so throughout these probes he remains in the span of the original vertex he occupied at the beginning on Step 2

1. The robber began Step 2 at vertex $v_i \in X$. At some point the cop would probe v_i, and hence receive a response at most $m-1$, locating v_i. At this point he must be on a thread from v_i to some v_j, which could be in M, X or I. She then continues probing through the rest of the midpoints of M and the set X, as per the above, and then repeats these probes in the same order. In order for him to reach another original vertex he will need to move m times, but he cannot try to reach another original vertex in M or X since if he tried to do so he would be located.

   Indeed if the robber's intended destination was in X, say v_j, then he would need m moves to reach v_j but the above probes would ensure that v_j was probed twice, at least one of which probes was after v_i had been probed (telling the cop which thread he was on) and at most $m-1$ probes apart, preventing him from leaving that thread. If his destination was in M then essentially the same occurs, since probing the midpoint would inform the cop he was heading to one of that pair, and then she can use an extra probe at one of them to locate him in that pair when he reached it. If he returns to v_i during the $m-1$ probes following when it was first probed she will get an answer divisible by m (possibly plus $m/2$ if she just probed a midpoint), and

hence know he is at some original vertex. Since he cannot have reach another other original vertex in this time, he must have returned to v_i, and thus this will locate him.

The only way he can avoid being located would be if he headed for a vertex in I. She will know this is the case if she completes her second set of probes without locating him. To force him to enter I she can then alternatingly probe v_i with a vertex from I until she has checked all of I. This prevents him from being able to return to X, and would locate him if he did not enter I, thus forcing him to enter I – again which will be detected by a probe returning a distance that is a multiple of m.

2. The robber began Step 2 at a vertex $v_i \in M$. The analysis proceeds in essentially a similar way – the cop will find a pair containing v_i when she probes the midpoint of the associated edge in M. She then proceeds with the same set of probes as before, which if he was heading to a vertex in $M \cup X$ locates him, or if he returns to v_i, and hence she concludes that he is on a thread heading to some vertex in I. Again she can then force him into I in a similar fashion.

The remaining case is that he started at some vertex in $v_i \in I$. In this case, the above analysis shows that he cannot reach another vertex in $M \cup X$, and hence he must either remain in the span of a vertex in I. The remaining strategy is treated very similarly in Steps 2 and 3, the cop can then repeat this set of $m-1$ probes, following each by a probe at a different vertex in I. Since she eliminates all the destinations every m steps, he cannot leave the span of I, and so must remain in the span of a single vertex in I throughout this process. Hence at some point she will probe v_i, and thus determine it. Now locating him is trivial, since he cannot remain on a single thread (or else he will be identified when she checks the original vertex

at the far end of it during the next $m-1$ probes), he cannot leave the thread through $M \cup X$ since every destination is checked every $m-1$ steps, and he cannot return to I or else she will get an answer consistent with being on an original vertex, and thus locate him. Hence he will be located, showing that $G^{1/m}$ is locatable if $m \geqslant n/2+1$. □

In fact it is clear from the proof that our result is in general slightly stronger than $n/2+1$, since the bound for m is phrased in terms of the actual structure of the graph. Specifically it is derived from the largest independent set and the size of the largest matching in the remainder. Hence for a general graph this may in fact give a bound smaller than $n/2+1$, but in the complete case it is tight.

CHAPTER 4

SUBDIVIDING INFINITE GRAPHS

The work in this chapter is all joint work with Sebastian Koch of Cambridge. For infinite graphs, defining the concept of locatability is a little more complicated. Recall that we define a graph to be *locatable* if the cop has a strategy that locates the robber in finite time, and *non-locatable* if the robber has a strategy that avoids being located indefinitely. Further recall that we have taken a slightly different view of the game from the standard, in which the robber does not occupy a single vertex but a set of vertices, and in which in order to locate him the cop must reduce his set of vertices to a singleton. This means that infinite graphs behave slightly differently to finite graphs. For example, although a finite graph with no edges is trivially locatable (the cop just needs to probe through the vertex set, as the robber cannot move), this question is a little more complicated even in the infinite countable case. The default view of the game would say that this graph was locatable, but with no defined location number, since for any starting position of the robber the cop will eventually probe that vertex, locating him, but for any k the robber can avoid being detected by time k by starting in any vertex not in the cop's first k probes. We find this analysis somewhat unsatisfying, and note that in our version this graph would be (we believe fairly) non-locatable, since a valid strategy for the robber is to continually give the response ∞ indicating that he is not in any vertex yet probed. At all times this would mean that there were still vertices that he could be in, so the cop would never locate him.

This illustrates some of the complexity of considering the game on infinite graphs. In this chapter we analyse the game more closely, showing that there are some simple obstructions to a graph being locatable. We then extend the work of [3] on finite graphs to show that, for any infinite graph not being trivially non-locatable by virtue of falling foul of one of these obstructions, there exists a suitable

subdivision function that turns it into a locatable graph. This is the main result that we shall work towards.

Theorem 17. Let G be an infinite graph with countably many vertices, each of finite degree, in finitely many components. Let

$$s : E(G) \to \mathbb{N}, xy \mapsto 2\max\{d(x), d(y)\} \tag{4.1}$$

Then $G^{1/s}$ is locatable.

This theorem obviously implies the main obstructions to a graph being locatable, namely the existence of a vertex of infinite degree, uncountably many vertices or infinitely many components. As these are preserved under taking subdivisions, the following lemmas suffice to show that no subdivision of these will be locatable either.

We begin with a simple lemma which shows that the robber can always hide for a certain amount of time in the neighbourhood of a vertex. This will yield the infinite degree result as a simple corollary.

Lemma 18. Let G be an finite graph that contains a vertex of degree Δ. Then the robber cannot be located before at least $\log \Delta$ probes have taken place.

Proof. We give a strategy that enables the robber to hide for at least $\log \Delta$ steps i.e. that ensures that for at least $\log \Delta$ steps he has at least two vertices he could be at. Let our vertex of degree Δ be v – we call it the *central vertex* – with closed neighbourhood $N[v] = N(v) \cup \{v\}$. At time 0 the robber could be anywhere on the graph, so he could be anywhere in $N[v]$. In fact we prove a slightly stronger result by showing that even if the robber restricts himself to $N[v]$ throughout the whole game he can remain undiscovered for at least $\log \Delta$ steps. This then implies the result we need by a simple argument, given for reference as Lemma 5 in Chapter 2.

We claim that at each timestep the robber can claim to be in a set at least half the size of the set he was in before until he is captured. We show this by an inductive argument. Based on the assumption that before the t^{th} timestep the robber is in a set of size $|M_{t-1}|$ inside $N[v]$, after the t^{th} probe he can be in a set of size at least $\frac{|M_{t-1}|}{2}$ inside $N[v]$. Let us assume that the cop probes p_t at time t, and let $d = d(p_t, v)$ be the distance between the probe vertex and the central vertex.

We first deal with the following special situation. If there exists $w \in M_{t-1}$ such that $d(p_t, w) = d$, the robber can claim to have been in v, and spread to all of $N[v]$ in his next step, which trivially satisfies the inductive claim. Hence we can disregard this possibility in what follows, and without loss of generality we may assume that the cop chooses a probing vertex such that, if possible, for all $w \in M_{t-1}$, we have $d(p_t, w) \neq d$. Thus by the triangle inequality, we have that for all $w \in M_{t-1}, d(p_t, w) \in \{d-1, d+1\}$.

We now consider separately three cases for the probing location p_t.

1. $p_t = v$: In this case, all vertices in M_{t-1} are at distance 1. The robber returns distance 1, and can in particular remain anywhere inside M_{t-1}. Thus $M_{t-1} \subseteq M_t$, and so $|M_t| \geqslant |M_{t-1}| \geqslant \frac{|M_{t-1}|}{2}$ as required.

2. $p_t \in N(v)$: By the above observation no vertices in M_{t-1} can be at distance 1 (and clearly only the probing vertex can be at distance 0), hence at least $|M_{t-1}| - 1$ of them are at distance 2. As $|M_{t-1}| \geqslant 2$, if the robber returns distance 2 the inductive step is satisfied again.

3. $p_t \in V(G) \setminus N[v]$: All vertices in $N[v]$ can only be at distances in $\{d-1, d, d+1\}$. By the above observation we may assume that there is none at distance d, hence M_{t-1} can be partitioned into two parts according to whether the vertices lie at distance $d-1$ or $d+1$. The robber can return whichever distance corresponds to the larger set, which will contain at least

half the vertices.

Thus the inductive step holds, and hence, as at each stage the robber can remain in a set of size at least half that he was in before it will take the cop at least $\log \Delta$ steps to locate him, regardless of her strategy. □

This gives rise to the following simple corollary.

Corollary 19. Let G be an infinite graph that contains a vertex of infinite degree. Then G is non-locatable.

Proof. Note that the proof of Lemma 18 is based on the idea that after every probe the robber can always hide in a set inside the neighbourhood of the central vertex of at least half the size of the set he was in before. In this case, inductively this means he can always hide in an infinite set, as he is in an infinite set initially and at every probe he can give an answer that preserves an infinite set for him to remain in. Thus the robber can hide indefinitely in this infinite neighbourhood. □

Lemma 20. Let G be an infinite graph with infinitely many components. Then G is non-locatable.

Proof. The proof is very similar to that in Lemma 19. As the graph has infinitely many components, the robber can form a winning strategy by always returning the distance ∞, claiming to be in the components which the cop has not yet probed. Since the cop can only probe finitely many vertices, and thus components, in finite time, the robber will always have some vertices that he could claim to be in, hence this is a winning strategy for the robber. □

These two results combine to give us the following theorem, which shows that any uncountable graph must be non-locatable.

Corollary 21. Let G be an infinite graph with uncountably many vertices. Then G is non-locatable.

Proof. Any graph with uncountably many vertices must either have a vertex of infinite degree or infinitely many components, hence the result follows from Lemmas 19 and 20. □

Because of Lemmas 19, 20 and Corollary 21, throughout the rest of this section we only consider infinite graphs on countably many vertices in finitely many components where all degrees are finite. In fact, as for the finite case, we may as well only consider connected graphs, since the cop can determine in fixed finite time which component the robber is on and then restrict to only playing on that component. We now wish to dismiss the possibility that given a fixed graph G there is always some number $m(G)$ such that if we subdivide each edge m times we obtain a locatable graph, as in the finite case. The following lemma shows that this is not true in general for infinite graphs.

Lemma 22. There exists an connected infinite graph G with countably many vertices and all vertex degrees finite such that for any integer m the graph $G^{1/m}$ is non-locatable.

Proof. We prove this by explicitly giving such a graph. Ideally we would just consider the graph consisting of disjoint unions of complete bipartite graphs $K_{t,t}$ for all $t \in \mathbb{N}_{\geqslant 4}$. It is clearly countably infinite, as it consists of a countable union of finite graphs, and all vertex degrees are finite as required. However it consists of infinitely many components. To circumvent this, consider a one-way infinite path with vertices $v_1, v_2, \ldots$, and attach to each v_i a pendant copy of $K_{i,i}$, in which each v_i is adjacent to only one vertex in the copy of $K_{i,i}$. This now only has one component, and is still an infinite graph with countably many vertices and all vertex degrees finite as required. This will be the graph G that we shall work with.

Now let $m \in \mathbb{N}$ be any integer. $G^{1/m}$ contains a copy of $H := K^{1/m}_{m+2,m+2}$ with only one edge from this to the rest of the graph. In Lemma 13 we show that

$K^{1/m}_{m+2,m+2}$ is not locatable (for $m \leqslant 2$ – for $m = 1$ it contains a copy of K_4). $G^{1/m}$ will contain such an induced graph, and thus by Lemma 7 $G^{1/m}$ here is non-locatable. As m was chosen arbitrarily the result follows. □

We now come to our main result, that for infinite graphs that do not satisfy one of the conditions above, it is possible to make them locatable by sufficiently – not necessarily uniformly – subdividing the edges. We shall show this for an explicit function for the subdivisions that has the advantage of giving a uniform upper bound on sufficient subdivision for a graph of bounded degree.

We shall prove Theorem 17 by presenting an explicit strategy for the cop, in which she always probes original vertices. Our proof is inductive, and in essence runs as follows. The cop picks a vertex $v_1 \in V(G)$ arbitrarily to probe first, receiving answer D_1. She then compares D_1 to $d(v_1)$, and adopts one of two approaches. If $D_1 \leqslant d(v_1)/2 + 1$ she can locate him before he can move through any original vertex other than v_1. If $D_1 > d(v_1)/2 + 1$ she carries out a sequence of probes that either locate him directly, or includes a probe at some other original vertex v_2 which gives a response less than D_1. She then repeats the argument starting from v_2, and continues until some probe at an original vertex v_i yields a distance at most $d(v_i)/2 + 1$. As distances must always be non-negative integers and there cannot be an infinite decreasing sequence of non-negative integers, this strategy must locate him.

We shall begin with a simple result, which shows that consecutive probes at endpoints of a thread either eliminate the possibility that the robber is on that thread or locate him. As we will make use of this fact at several points in our cop strategy we present it as a separate Lemma.

Lemma 23. Let G, s and $G^{1/s}$ be as in the statement of Theorem 17. Let v and w be original neighbours in $G^{1/s}$. Then, if the cop makes consecutive probes at v and w and the robber is on the $v \cdots w$ thread when the second probe is carried out, this

locates the robber.

Proof. Without loss of generality let the first probe be at v and the second be at w. Let the distance returned by the probes at v and w be D_v and D_w respectively. Let r be the unique vertex at distance D_w from w along the $v \cdots w$ thread – note that this is unique by the choice of s. The only way that the cop could not then conclude the robber was at r would be if there was some other possible location r' at distance D_w from w for the robber. Hence r and r' must have some vertex inside their respective closed neighbourhood at distance D_v from v. We show by contradiction that no such r' can exist.

First, we eliminate one straightforward situation. If either v or w is of degree 1 then the probe at that vertex locates the robber immediately. Hence we may assume that they are both of degree at least 2. As r' is not on the $v \cdots w$ thread there must be another original vertex u that either lies on a shortest path from r' to w or from r' to v; we assume the former, and note that the argument is symmetrical in the other case. We illustrate this case in Figure 4.1 to make it easier to follow the following calculations. We have

$$\begin{aligned} d(v, r') + d(r', w) &= d(v, u) + d(u, w) \\ &\geqslant 2d(v) + 2d(w) \\ &\geqslant 2\max\{d(v), d(w)\} + 4. \end{aligned}$$

Note moreover that $d(v, r') \leqslant D_v + 1$ and $d(r', w) \leqslant D_w + 1$. Hence

$$\begin{aligned} D_v + D_w &\geqslant d(v, r') + d(r', w) - 2 \\ &\geqslant 2\max\{d(v), d(w)\} + 2. \end{aligned}$$

But by the fact that r is at distance D_w from w along the $v \cdots w$ thread and a vertex from its closed neighbourhood is at distance D_v from v, we deduce that $D_v + D_w \leqslant 2\max\{d(v), d(w)\} + 1$. Combining these inequalities gives $2\max\{d(v), d(w)\} + 1 \geqslant 2\max\{d(v), d(w)\} + 2$, yielding a contradiction.

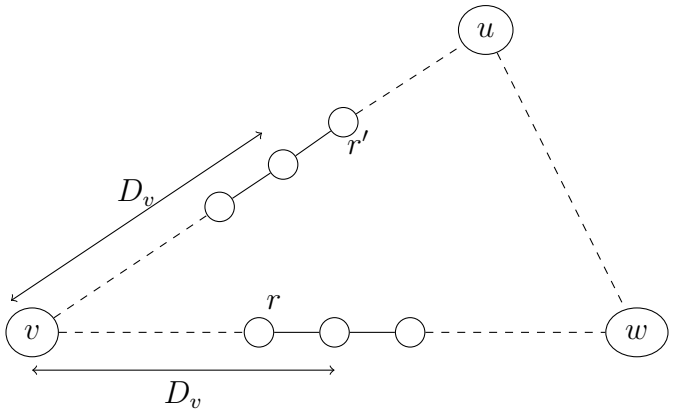


Figure 4.1: r and a possible location for r'

□

We now proceed by outlining the winning strategy for the cop and begin by addressing the situation where she probes some original vertex v and receives and answer which is at most $d(v)/2 + 1$. We present this as a separate lemma, in order to refer to it in the proof of the main theorem.

Lemma 24. Let G, s and $G^{1/s}$ be as in the statement of Theorem 17. If a probe at any vertex v of G returns an answer at most $d(v)/2 + 1$, then the cop can locate the robber before he passes through another original vertex.

Proof. Let us refer to the probe which the cop carried out at v, with associated distance at most $d(v)/2 + 1$, as the 0^{th} probe, so that the following one will be the first. Let k be the degree of vertex v to ease notation, and denote the k original neighbours of v as $b_1, \ldots, b_k$. Let m be such that the robber's initial location is on the $v \cdots b_m$ thread. The cop's strategy has her carry out probes working through the original neighbours in some fixed order (say according to their index), some of

them followed by another probe at v. The result of the probe at v determines how many original neighbours she probes afterwards before probing v again. If the robber gives an answer of 1 or 2 she only probes a single original neighbour before returning to v, otherwise she probes two original neighbours. We claim that this strategy locates the robber if he remains on the $v \cdots b_m$ thread and show that he cannot leave this thread without being located.

To show that the robber is located if he remains on the thread, note that eventually the cop will have probed every original neighbour of v. Further, as she probes v at least at every third step, this probe is either directly followed or directly preceded by a probe at v. Hence by Lemma 23 if the robber remains on the $v \cdots b_m$ thread he is eventually caught.

Next, we show that the robber cannot leave the $v \cdots b_m$ thread. To do so he would have to either pass through v or b_m. In the former case, assume he is in v at time t. Note that if the robber is within 2 steps of v the cop probes v at every other step. Hence she must probe v either at time t or time $t-1$. If the cop probes v at time t the robber is located. Thus we assume that she probes v at time $t-1$ and thus received answer 1. Let b_i be the original neighbour of v she probes at time t. Thus by Lemma 23 the robber is located by her probe at b_i.

To see that the robber cannot leave the $v \cdots b_m$ thread by passing through b_m requires us to count the total number of probes made. The calculations differ only slightly depending on whether k is even or odd. We present those when k is odd, but the case when k is even is essentially identical. Let $k = 2q+1$. Recall that the robber starts at distance at most $\lfloor d(v)/2 + 1 \rfloor = q+1$ away from v. To simplify the calculation assume that $k \geqslant 5$ so that this initial distance is at least 3. The remaining small cases are easily verified by hand. First the cop carries out q sets of 3 probes, each consisting of a pair of original vertices followed by a probe at v. This takes $3q$ turns. After this and his following move the robber can at most be at

distance $4q+2$ from v. But as $d(v) = k = 2q+1$, the $v \cdots b_m$ thread has length at least $4q+2$. Hence the robber has possibly reached b_m, but not passed through. The cop only has one remaining original vertex to check, namely b_k, which she probes on her next and final turn. The robber cannot leave the $v \cdots b_m$ thread during this process, and so is located by this strategy using Lemma 23 as soon as the two endpoints have been probed consecutively.

□

We now proceed to the proof of Theorem 17. We follow the proof outline given above, detailing a strategy that makes use of Lemma 24 as a sub-strategy if any probe finds that the robber is close enough to the probed vertex.

Proof of Theorem 17. Consider the following strategy for the cop, consisting of always probing original vertices, starting from an original vertex v_1 picked arbitrarily. Let the distance returned be D_1. If D_1 is at most $d(v_1)/2+1$ then the cop has a winning strategy by Lemma 24, so we may assume that $D_1 > d(v_1)/2+1$. We give a sequence of probes that fall into one of the following situations.

1. A probe at another original vertex b_i that gives a response at most $d(b_i)/2+1$.
2. A probe at another original vertex that gives a response less than D_1. Labelling this vertex as v_2 and the distance from it as D_2 the cop begins her strategy anew from v_2.

This suffices to locate the robber since the cop cannot find an infinite sequence $v_1, v_2, \ldots$ with associated decreasing distances $D_1 > D_2 > \ldots$ and hence the second situation cannot arise infinitely often. Thus she must at some point encounter the first situation. When this happens, the cop switches to the strategy outlined in Lemma 24 to locate the robber directly.

The cop's strategy begins by partitioning the original neighbourhood of v_1 into two sets. Let A_1 comprise the original neighbours that are at most distance D_1

away from v_1, and B_1 comprise the original neighbours that are further than D_1 from v_1. As the robber begins at distance D_1 from v_1, then one of the two following cases must hold.

(a) There exists some $a \in A_1$ such that a lies on a shortest path from v_1 to the robber. Hence $D_1 = d(v_1, a) + d(a, r)$, where r is the position of the robber.

(b) The robber is on a thread between v_1 and some vertex in B_1.

The cop begins by investigating possibility (a). To do this, she probes the vertices in A_1 in any order. This takes at most $d(v_1)$ steps, and since a lies on a shortest path from v_1 to the robber's initial position, a is initially at least $2d(v_1)$ closer to the robber than v_1. Hence when she probes a she receives a distance that is at most $D_1 + d(v_1) - d(v_1, a) < D_1$, satisfying condition (ii) above. Note that it is possible that some vertex $a' \in A_1$ probed earlier already gives a response less than D_1, so this strategy may not find a itself, but this does not matter for our analysis.

Thus having carried out these probes the cop knows that initially case (b) occurred and the robber was on a thread between v_1 and one of its original neighbours in B_1. Let k be the size of B_1, and let us label the vertices in B_1 as $\{b_1, \ldots, b_k\}$. Without loss of generality we may assume that $k = d(v_1)$ i.e. that $A_1 = \phi$. The cop's strategy now has her carry out sets of probes, which we denote by $P_1, \ldots, P_{k+1}$. Each of the P_i consists of $(k+1)$ probes, k at the vertices in B_1 in order according to their index, with v_1 inserted in the i^{th} place. Hence P_1 consists of probing $v_1, b_1, \ldots, b_k$, P_2 is $b_1, v_1, b_2, \ldots, b_k$ etc.

Note that, by construction, if the cop carries out these sets of probes then the following two conditions are fulfilled – firstly the cop probes each vertex in $\{v_1\} \cup B_1$ at least at every $(k+2)^{\text{nd}}$ step, and secondly that for every $b \in B_1$ at some point she probes v_1 and then immediately b. The former ensures that, if the robber passed through either v_1 or some $b \in B_1$, he would then be located by

Lemma 24 as he would have to be within distance $k/2 + 1$ of it either when it was last probed, or when it next is. The second condition ensures that, if the robber remains inside a single thread, even when far enough away from all original vertices to not allow the cop to apply Lemma 24, he will be located, using Lemma 23. □

CHAPTER 5

LOCATION NUMBER AND MAXIMUM DEGREE

The work in this chapter is all joint work with Sebastian Koch of Cambridge. In her original paper, Seager [10] defined the *location number* of a locatable graph as the minimum number of probes needed to guarantee locating the robber. In Chapter 1 we outlined an algorithm that determines this for any graph, but the question remains of whether or not in general it can be bounded by some natural graph parameters. In this chapter we seek bounds in terms of the maximum degree of the graph (denoted $\Delta(G)$ or just Δ where the underlying graph is clear), in other words whether we can say that a locatable graph with maximum degree Δ always takes at least $f(\Delta)$ steps to locate the robber, and, at most $g(\Delta)$ steps. We have already seen that any graph has location number at least $\log \Delta$ in Chapter 4 as Lemma 18, our main result in this chapter is that this looks to be the correct lower bound. We show this by exhibiting graphs with location number $\log \Delta + \log(\log \Delta) + 2$, which is clearly dominated by the $\log \Delta$ term. We also show that no upper bound is possible in terms of the maximum degree by exhibiting locatable graphs with maximum degree 3 that have unbounded location number. As an aside, we also give an example of a graph with maximum degree three which is non-locatable, hence showing that it is not even possible to tell purely from knowing about the maximum degree of a graph whether or not it is locatable.

We shall begin with the lower bound, and Theorem 25 showing that $f(\Delta) \leqslant \log \Delta + \log(\log \Delta) + 2$, where both logarithms are in base 2. We then investigate the family of truncated binary trees, which have maximum degree 3, and show that they are non-locatable, but that sufficiently subdivided members of this family are locatable, still have maximum degree 3 and have arbitrarily large location numbers. We begin with the first lower bound result.

Theorem 25. For infinitely many Δ there exist graphs with a vertex of degree Δ with locatability number at most $\log\Delta + \log(\log\Delta) + 2$.

Proof. We shall prove this by a constructive example, which shall consist of a star with Δ leaves, augmented by adding vertices and edges until a graph is reached which is locatable in at most $\log\Delta + \log(\log\Delta) + 2$ steps.

Intuitively we want to add $\log\Delta$ new vertices, and join them to the leaves of the star according to the binary expansions of the indicies of the leaves. Then, provided the robber stayed still, the cop could probe these $\log\Delta$ new vertices, and by analysing which gave her distance 1 she could work out the binary expansion of the leaf the robber was hiding in, and thus locate him. This will not however work directly as in practice the robber can move, which requires us to modify this technique a little.

In practice, we construct our graph as follows. Let $\Delta = 2^{2^k}$ for some $k \in \mathbb{N}$. We choose this Δ so that all the numbers in this proof shall be integers. Begin with a star of degree Δ around a *central vertex* v with *leaves* $l_0, \ldots, l_{\Delta-1}$. Add $\log\Delta$ new isolated vertices, which we call the *first control vertices*, labelled $c_1, \ldots, c_{\log\Delta}$. Connect the first control vertices to the leaves according to the binary expansions of the indices of the leaves, i.e. for all $0 \leqslant i \leqslant \Delta - 1$ and $1 \leqslant j \leqslant \log\Delta$ connect v_i to c_j if the j^{th} position in the binary expansion of i is a 1. Now repeat this process by creating $k = \log(\log\Delta)$ new isolated vertices, which we call the *second control vertices*, labelled $d_1, \ldots, d_k$, and connect the second control vertices to the first control vertices according to the latter's binary expansions. Thus for all $0 \leqslant i \leqslant \log\Delta$ and $1 \leqslant j \leqslant \log\Delta$ connect c_i to d_j if the j^{th} position in the binary expansion of $i - 1$[1] is a 1. We then subdivide each edge $4\log\Delta$ times. For illustration, an example of this graph in the case $k = 2$ is shown in 5.1. We say that two original vertices are *connected directly* if they were neighbours in the

[1]The additional factor of -1 here is for technical reasons since really you want these indices to run from 0 to $\log\Delta - 1$.

unsubdivided graph i.e. if the shortest path between them does not pass through any other original vertex.

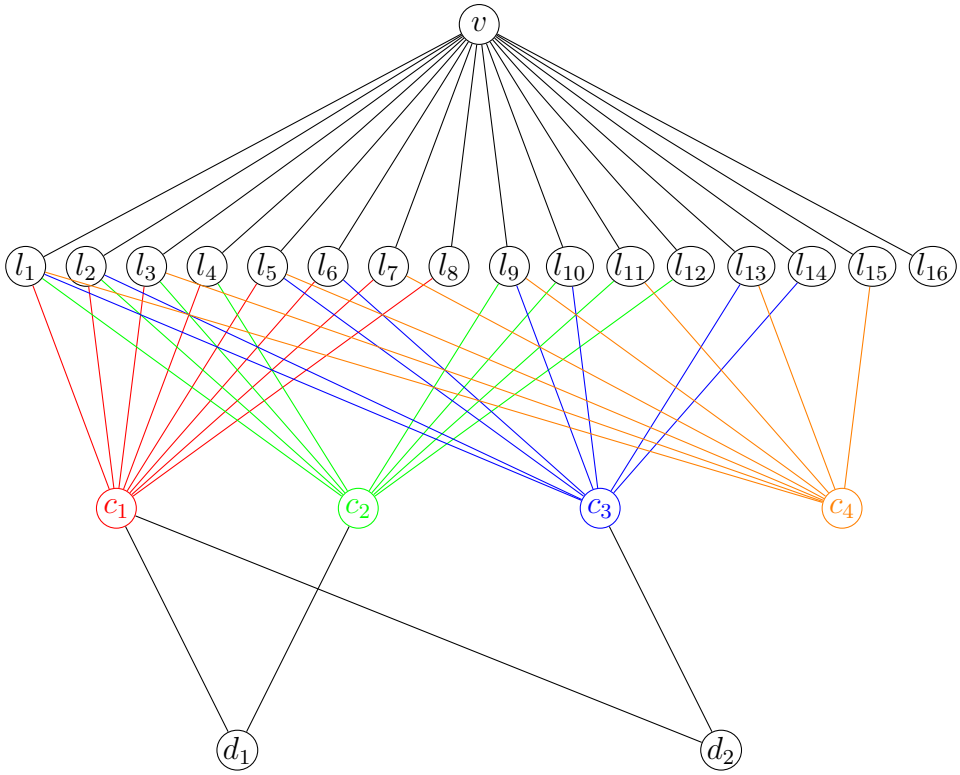


Figure 5.1: Logarithmically-locatable graph with a high degree vertex

Having described the graph, our next task is to present the strategy for the cop. We start with a sketch of the idea. The cop begins by probing v to establish where the robber begins, and then tries to locate him before he can move too much. If he begins near the central vertex then she does this by probing the first control vertices – this shall identify which $v \cdots l_i$ thread he is on, and thus locate him. If he begins near a leaf then she first again probes the first control vertices to determine which leaf it is, checks to see if he is above or below it and, if the latter, probes the second control vertices to locate him. The case where he begins near a first control vertex is essentially similar, and the final case where he begins near a second control vertex is even simpler since she can probe all the first and second control vertices before he can leave the span of the second control vertex he began near.

Now we describe the strategy in detail. As outlined in the previous paragraph, the cop begins by probing the vertex v and her strategy splits into four cases according to the initial response d_1 of the robber.

1. $d_1 \leqslant 2\log\Delta$. Note that provided the cop does not take more than $2\log\Delta$ steps in this case the robber cannot pass through any of the leaves. As her initial probe was the central vertex he cannot be there, so he must be on some $v \cdots l_i$ thread. It will suffice for her to find out the value of l_i, thereafter locating him is trivial. She does this by probing the control vertices in $\log\Delta$ steps. Note that the central vertex is at distance $8\log\Delta$ from the first control vertices, and no other vertex in the span of it is, so if he returns to v he will return this response and thus be located. Further as she has uses $\log\Delta$ probes, the robber cannot have passed through a leaf, hence he remains in the same span throughout this first set of probes. We now claim that a probe at a control vertex c_j reveals whether or not c_j is connected directly to l_i or not, since in the former case it will give a response smaller than $8\log\Delta$, and in the latter case it will give a response larger than $8\log\Delta$. Hence having completed the $\log\Delta$ probes at the first control vertices she knows the value of l_i. If necessary she can use one final probe at v now to locate him, in total using at most $\log\Delta + 2$ probes as required.

2. $2\log\Delta < d_1 \leqslant 6\log\Delta$. In this case the robber is in the span of one of the leaves, and within $2\log\Delta$ of that leaf. We denote the leaf he is in the span of by l_i. Again so long as the cop doesn't take more than $2\log\Delta$ steps to locate him he must thus remain in the span of l_i. First the cop uses $\log\Delta$ steps to identify l_i as in the above case by probing the first control vertices, since again responses less than $8\log\Delta$ correspond to the probed first control vertex being connected directly to l_i and responses greater than $8\log\Delta$ correspond to the probed first control vertex not being connected directly to l_i. Hence after

these $\log \Delta$ probes she knows which leaf is l_i. She now probes v once to discover if he is on a $v \cdots l_i$ path (signified by a response less than $4 \log \Delta$ and locating him), at l_i (signified by a response of $4 \log \Delta$ and again locating him) or on some $l_i \cdots c_j$ thread (signified by a response of more than $4 \log \Delta$). In this final case she just needs to determine the value of c_j, which she does by probing the second control vertices in the same way as she probed the first control vertices to determine l_i. Again during this process he cannot return to l_i, or else she will detect this by a response of either $8 \log \Delta$ or $16 \log \Delta$ (depending on the length of the path beteen the probed second control vertex and l_i). Otherwise responses of less than $8 \log \Delta$ mean the probed second control vertex is connected directly to c_j, all other responses mean the probed second control vertex is not connected directly to c_j. Hence at the end of these $\log(\log \Delta)$ probes she will know c_j, and hence locate him. At most this takes $\log \Delta + \log(\log \Delta) + 2$ probes as required.

3. $6 \log \Delta < d_1 \leqslant 10 \log \Delta$. In this case the robber is in the span of one of the first control vertices, which we denote as c_j. This proceeds simiarly to the above, the cop first probes the second control vertices, using $\log(\log \Delta)$ probes, to determine the value of c_j. She then probes v to determine if he is on a $l_i \cdots c_j$ thread or a $c_j \cdots d_i$ thread. In the former case she then probes all the first control vertices to determine the value of l_i, and thus locates him, in the latter case she probes all the second control vertices to see which gives a response of less than $4 \log \Delta$, which thus determines the value of d_i and locates him. At most this would thus take $\log \Delta + \log(\log \Delta) + 2$ probes as required.

4. $10 \log \Delta < d_1$. This means that the robber is initially in the span of one of the second control vertices. In this case the cop can first probe the second control vertices to determine which it is, which will be signified by a response of at

most $4 \log \Delta$. At this point she knows he must be on some $c_i \cdots d_j$ thread, and she knows d_j. Now she can probe all the first control vertices to discover c_i and thus locate him, noting that he cannot leave this thread through c_i as he doesn't have enough time to reach it, and cannot do so through d_j since doing so would return a response of a multiple of $4 \log \Delta$, thus revealing he was at d_j and locating him. At most this would thus take $\log \Delta + \log(\log \Delta) + 1$ probes, also fulfilling our requirements.

□

Lemmas 18 and 25 together give the order of magnitude of a general lower bound for locatability number for finite graphs, based on the maximum degree. At the same time, they leave us with an open problem, namely to determine the precise value of $f(\Delta)$. We believe that the correct answer is actually $\log \Delta$, since this is the answer in the above if the robber started close to the central vertex, which is the vertex of highest degree. It is possible that a better construction exists that could be put into place below the leaves that would allow him to be located more quickly if he was far from the central vertex.

Next, we show that, in contrast to the lower bound just established, there is no upper bound for the locatability number in term of the maximum degree Δ. We prove this using subdivided binary trees. A *binary tree*, denoted T_k, is a rooted tree with each vertex having 2 children in the layer below until you reach the leaves at distance k from the root. It is naturally partitioned into layers, with the t^{th} layer (denoted L_t) consisting of the vertices at distance t from the root – hence T_k contains $k+1$ layers, $\{L_0, L_1, \ldots, L_k\}$. We include an illustration of T_3 in Figure 5.2a. It is a simple exercise to check that a binary tree with at least 5 layers is actually non-locatable, as we show in Chapter 2, Lemma 6. However, note that subdividing such a tree shall result in a tree with the same maximum degree i.e. maximum degree 3, and shall eventually result in a locatable graph (this being the

major motivation of the work of [3], and equally our results in Chapter 3). Hence a sufficiently subdivided binary tree shall still have maximum degree 3 and be locatable. However such trees may have arbitrarily large location number as the following result shows.

Lemma 26. The graph $T_k^{1/m}$ has location number at least k for all values of m.

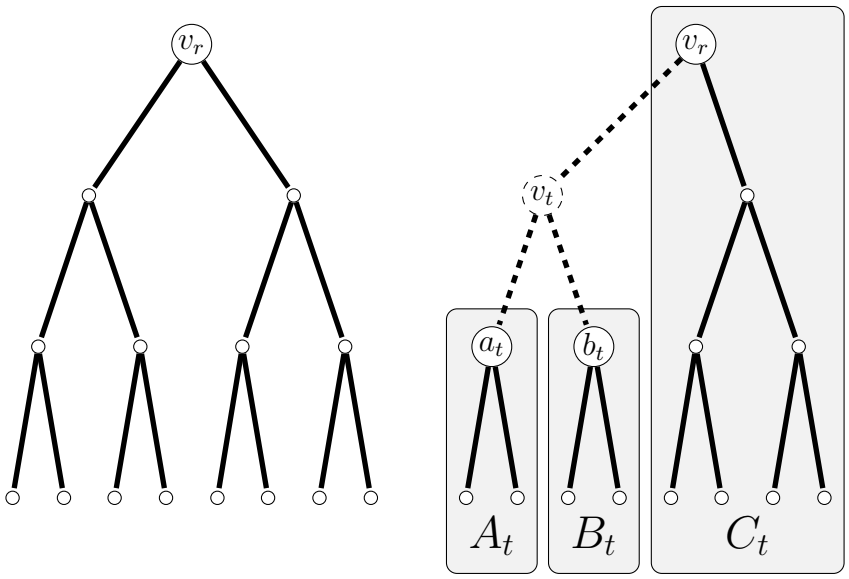
Proof. We present a strategy for the robber, in which he hides in the leaves of the graph. Let the *shadow* of a vertex be the set of leaves that are descended from it – thus formally a leaf w is in the shadow of v if v lies on the (unique) path from w to the root v_r. Further, with slight abuse of notation, let the t^{th} layer of the subdivided tree still consist of the original vertices that were at distance t from the root before it was subdivided – so now they are those vertices at distance mt from the root. We claim that, inductively, in response to the cop's t^{th} probe, the robber can claim to be anywhere in the shadow of a t^{th} level vertex for $1 \leqslant t < k$. Thus as these shadows all contain at least two elements for $t < k$ the robber survives the first $k-1$ probes, and the locatability number of $T_k^{1/m}$ is at least k.

For the cop's first probe let a_1 and b_1 be the children of the root v_r. Let A_1 and B_1 be vertex sets defined as follows: A_1, respectively B_1, is the vertex set containing a_1, respectively b_1, and all of its descendants. This partitions the vertex set of $T_k^{1/m}$ into $\{v_r\} \cup A_1 \cup B_1$. Note that the set of leaves in A_1 comprises the shadow of a_1, and every vertex in this set is equidistant to any fixed vertex in $\{v_r\} \cup B_1$. The same holds with b_1 in the place of a_1 and A_1 and B_1 reversed, by symmetry. Hence the robber can claim to be in the shadow of a_1 if the cop's first prove is in $\{v_r\} \cup B_1$, or in the shadow of b_1 if the cop's first probe is in A_1. This fulfils the base case of the induction.

Now assume that the claim was true at time $t < k$, and consider the $(t+1)^{\text{st}}$ probe. Let $v_t \in L_t$ be the vertex whose shadow the robber claimed to have been in at time t and let a_t and b_t be v_t's children. Consider the components of the graph

inducted by deleting the vertex v_t. There are at most three components, one for each child of v_t and one corresponding to the parent (which does not exist if v_t was the root). Let us label them A_t, B_t and C_t as illustrated in Figure 5.2b, where A_t and B_t are the components containing a_t and b_t respectively. Note that every vertex in the shadow of a_t is equidistant to any fixed vertex in $\{v_t\} \cup B_t \cup C_t$, and every vertex in the shadow of b_t is equidistant to any fixed vertex in A_t.

Having described this perspective on the structure of the tree we next outline the robber's strategy – similarly to the base case, if the cop's next probe is in $\{v_t\} \cup B_t \cup C_t$ he claims to be in the shadow of a_t, otherwise he claims to be in the shadow of b_t. This fulfils the inductive condition, and thus completes the proof. □



a) T_2^4 b) Example partition

Figure 5.2: Binary tree with 4 layers

Note that this result held for all values of m, and hence it holds for m being very large. But m being very large essentially restricts the robber to not being able to move – certainly if m is much larger than k then the robber must remain in the span of a single original vertex throughout these k probes. A little case analysis

shows that this actually results in him being located within the k probes, and hence the location number for $T_k^{1/m}$ is k for large enough m. Hence it is not possible to give an upper bound for the location number in terms of the maximum degree, since graphs exist with maximum degree 3 of unbounded location number.

CHAPTER 6

SUBGRAPH CHARACTERISATIONS

The work in this chapter is all joint work with Sebastian Koch of Cambridge. In her inaugural paper on the Robber Locating Game, [10], Seager shows that any graph containing K_4 as a subgraph, or $K_{3,3}$ as an induced subgraph, is non-locatable. She also asked if a forbidden subgraph characterisation of locatable graphs might be possible, i.e. if being locatable might be *hereditary* (preserved under taking induced subgraphs) or even *monotone* (preserved under taking any subgraphs). Our result in this chapter is that surprisingly it is not even hereditary.

Theorem 27. The graph property of being locatable is not hereditary.

We shall show this by an explicit example. We shall include the *no-backtrack* condition in this chapter since Seager included it in her original paper, to most accurately answer the question posed. This condition states that the robber cannot move to the vertex just probed by the cop, which shall be used in Lemma 29 to limit the robber's movements. The question of whether or not locatability might be hereditary remains open if you drop this condition.

Consider the pair of graphs shown in Figure 6.1. The first, which we call the *double-net*, is a C_3 with two pendant edges on each vertex. We label the vertices on the cycle a, b and c. The second, which we call the *rooted double-net*, is obtained by taking a double-net and linking the two leaves that were adjacent to a by a path of length 2 through a new vertex, which we label r. Clearly the double-net is an induced subgraph of the rooted double-net, so the following two lemmas suffice to show the result.

Lemma 28. The double-net is non-locatable.

Proof. To prove this, we describe a winning strategy for the robber on the double-net which restricts him to one of three pairs of response/movement sets.

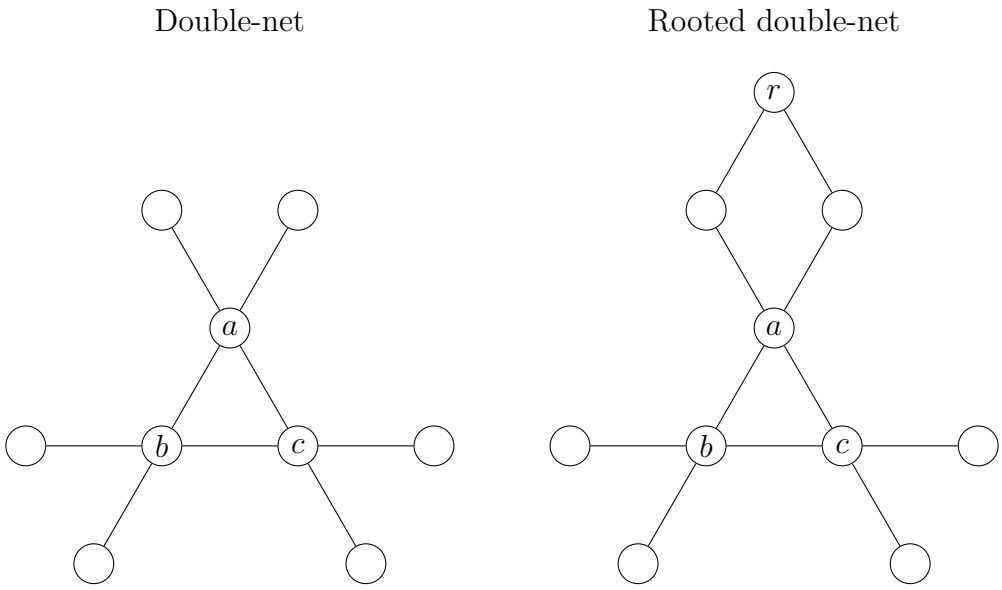


Figure 6.1: Double-net and Rooted double-net

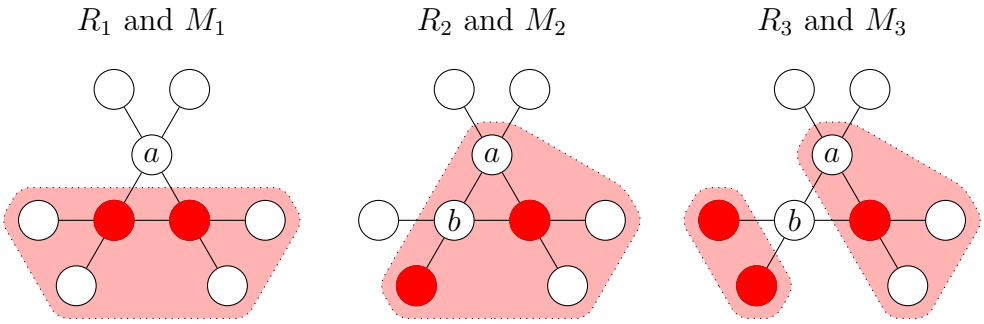


Figure 6.2: Response and movements sets for Lemma 28

This suffices to show that the double-net is non-locatable by Lemma 4 in Chapter 2. Consider the three sets which we illustrate in Figure 6.2. In each case we show the response sets with solidly coloured vertices, and the resulting movement set with a shaded background. As will be outlined in the strategy below the set R_3 will be returned following a probe at b, hence $b \notin M_3$ by the no backtrack condition. Similarly R_1 may be a response following a probe at a, so again $a \notin M_1$. By Lemma 3 we may assume that the robber begins in M_1. We shall give a strategy that ensures that he can always remain in one of these three sets, or a set equivalent to one of them by symmetry. At points the robber may have the option instead to

move to a superset of one of these, but by Lemma 4 we may ignore this and restrict him to only moving between these sets. Our strategy splits as follows according to the movement set he is in when the cop makes a probe.

1. M_1: If the cop probes a or a leaf adjacent to a the robber can return distance 1 or 2 respectively, giving response set R_1 again and so remaining in M_1. By symmetry the remaining cases are a probe at b, or a leaf adjacent to b. Following a probe at a leaf adjacent to b the robber can respond 2, giving a response set equivalent to R_2 and hence moving to M_2. Following a probe at b the robber can respond 1, giving response set R_3 and hence moving to M_3.

2. M_2: If the cop's next probe is in any leaf, or in one of $\{a, b, c\}$, the robber can return distance 2 or 1 respectively, corresponding to a response set equivalent to R_1 and thus moving to M_1.

3. M_3: If the cop's next probe is at a, or a leaf adjacent to a, then the robber can return distance 2 or 3 respectively. This doesn't quite give response R_1, instead the response set consists of the leaves of b and c, but it still allows the robber to move to M_1, so is equivalent for our purposes. If the cop's next probe is at b, or a leaf adjacent to b, then the robber can return distance 1 or 2 respectively. This returns a response set containing a set equivalent to R_1, and so he can move to M_1. If her next probe is at c then he can return distance 1, returning a set equivalent to R_3 and so remaining in M_3. Finally if her probe is at a leaf adjacent to c he can return distance 2, corresponding to a response set equivalent to R_2 and moving to M_2.

Since this shows there are a finite number of response sets and movement sets, such that from any movement set for any choice of probe vertex he can move to another, then by Corollary 4 the double-net is non-locatable. □

Lemma 29. The rooted double-net is locatable.

Proof. We show this by describing a winning strategy for the cop on the rooted double-net. Let her first three probes be the two leaves adjacent to b, followed by a probe at b itself. By the no-backtrack condition, and the fact that b is the unique vertex at distance 1 from its adjacent leaves, this ensures that after this third probe he cannot be in any of these three probed vertices. Next she probes r. The robber cannot respond with 0, 2 or 3 as each of these correspond to a unique vertex in the remaining set that he occupies. Hence he must return either distance 1 or 4. In either case he then moves to an induced path on three vertices, so can be located in a single additional probe. □

This shows that a general forbidden subgraph characterisation of locatable graphs is not possible.

CHAPTER 7

LOCATABILITY AND COLOURABILITY

The work in this chapter is all joint work with Sebastian Koch of Cambridge. In this section we consider the relationship between locatability and chromatic numbers. Recall that the *chromatic number* of a graph is the minimal $k \in \mathbb{N}$ such that there exists a partition of $V(G)$ into k sets with no edge contained in any part. It is referred to as the chromatic number as it can also be phrased as follows: We call a function $c : V(G) \to [l]$ a *colouring function* with l colours if no edge receives the same colour for both of its end points. We say a graph is *l-colourable* if it is possible to find a colouring function with l colours, and then we define the chromatic number of a graph as the minimal k for which it is k-colourable. We shall show that every locatable graph has chromatic number at most 4 (although clearly it can also be less), and that this result is tight by exhibiting a locatable graph that is not 3-colourable.

As in Chapter 6 we include the no-backtrack condition in this chapter, since this is work that was done early on when the main extant paper included this condition. If you drop this condition then these results can be improved to be essentially trivial, as we shall summarise at the end of this chapter.

We begin with the result that every locatable graph is 4-colourable.

Theorem 30. Every locatable graph is 4-colourable.

We shall show this by proving that every graph that is not 4-colourable must be non-locatable by showing that it must contain a small subgraph which the robber can hide in. Specifically we shall show that it must contain an induced subgraph with minimum degree 4, and that the robber can avoid being located while remaining in this subgraph.

Lemma 31. Let G be a graph that is not 4-colourable. Then G contains an induced subgraph H such that H has minimum degree at least 4.

Proof. Let H be a minimal induced subgraph of G such that H is not 4-colourable. We show by contradiction that H has minimum degree at least 4, by assuming that $\exists x \in H$ such that $\deg_x(H) < 4$. Then by the choice of H, $H \setminus \{x\}$ is 4-colourable. But then as x has at most 3 neighbours, then it is also possible to colour x using one of these 4 colours, and hence H would be 4-colourable, a contradiction. Hence no such x exists, and hence H is an induced subgraph of G with minimum degree at least 4. □

We now introduce the concept of a *hideout graph*. A hideout graph H is a graph such that any graph G containing H as a subgraph (not necessarily induced) is non-locatable. For example, Seager shows in [10] as Proposition 2.3 that K_4 is a hideout graph. We shall now show that any graph with minimum degree 4 is a hideout graph, which shall suffice to essentially complete the proof.

Lemma 32. Let H be a graph with minimum degree 4. Then H is a hideout graph.

Proof. Let G be a graph containing a copy of H. We present an inductive proof in which the robber restricts himself to remaining within H and can still avoid being located. Assume that he has survived until time $t-1$, hence before the t^{th} probe he is in a movement set $M_{t-1} \subset H$. Let $x \in M_{t-1}$ be any vertex in this set. and let p_{t-1} be the cop's probe at time $t-1$. Since $x \in H$ it has degree at least 4, so $N[x] \setminus \{p_{t-1}\}$ is a subset of M_{t-1} of diameter at most 2 with at least 4 vertices in it. Denote this set by M_x. Let p_t be the cop's probe at time t, and let d be the distance from p_t to M_x. Then every vertex in M_x is at either distance d, $d+1$ or $d+2$, hence this probe partitions M_x into at most 3 sets. As it has 4 members, at least 2 are equidistant, giving the robber a response he can return that avoids being located at time t. Hence by induction he can avoid being located indefinitely. □

With these results in mind, the proof of Theorem 30 is straightforward.

Proof. (Proof of Theorem 30): We prove this by contradiction. Let G be a graph that is not 4-colourable. Then by Lemma 31 it contains an induced subgraph H with minimum degree 4. By Lemma 32 H is a hideout graph, and thus G is non-locatable. Thus any locatable graph must be 4-colourable. □

This result is tight in that there exist graphs which are locatable and have chromatic number 4, so not all locatable graphs are 3-colourable. To show this we give an explicit example in Figure 7.1. This graph is essentially a subdivided K_4, except that instead of replacing edge with paths we have replaced them with chains of what we call *diamonds*, copies of C_4 with an extra edge across the middle. These chains, each of which consists of 8 diamonds, then has an extra edge added in the middle linking the 4th and 5th diamonds. We refer to the original vertices, i.e. those of degree 6, as the *corner vertices*.

These chains are what leads to this graph being both locatable and not 3-colourable. First, we shall show that if the cop knows that the robber is on a certain chain then she can 'push' him along it, forcing him to leave it through either end. It will follow immediately that if she pushes him towards the middle edge of a chain then she can locate him. Then to show that the graph is locatable all we need to show is that she can find out which chain he is on, which is straightforward. Finally to show that it's not 3-colourable we shall use the fact that if the graph is 3-coloured then the end vertices of a chain must be the same colour, and show that no such colouring can exist. We begin with the result on the cop being able to push him down chains.

Lemma 33. If the robber is known to be inside a specific chain of diamonds, the cop can force him to exit the chain through either end. Further, if this chain ends in a pendant edge then this will locate him.

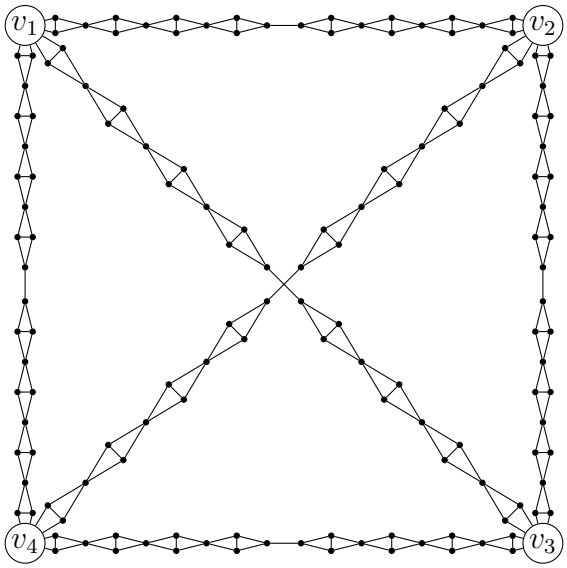


Figure 7.1: K_4 with edges replaced by pairs of chains of diamonds, linked by a single edge

Proof. Let $D_1, \ldots, D_k$ be k diamond subgraphs such that they form a chain. Let $v_0, \ldots, v_k$ be the tip vertices of these diamonds, so that $v_i = D_i \cap D_{i+1}$ for $1 \leqslant i < k$, and v_0, v_k are the remaining tip vertices of D_1 and D_k respectively. We shall give an explicit strategy for the cop that allows her to force him to leave the chain through her choice of v_0 or v_k.

First let the cop probe one endvertex of the chain. If he returns an even distance this means he is occupying a (unique) tip vertex, and hence locates him. Thus be must return an odd distance, $2i - 1$, which means he is in the two girdle vertices in D_i. We therefore can begin from the position where the robber occupies D_i. It suffices to show that the cop can force him to move to D_{i+1}, since repeating this will force him into v_k, fulfilling one requirement, and by symmetry this can be adapted to instead force him into v_0.

To ease notation we label the vertices of D_i and D_{i+1} as per Figure 7.2. The cop's strategy now runs as follows. First she probes v_{i-1}, forcing the robber to return distance 1 to avoid being located. Hence his first response set is $\{a, b\}$, and his next movement set is thus $\{a, b, v_i\}$. She then probes a, again forcing him to

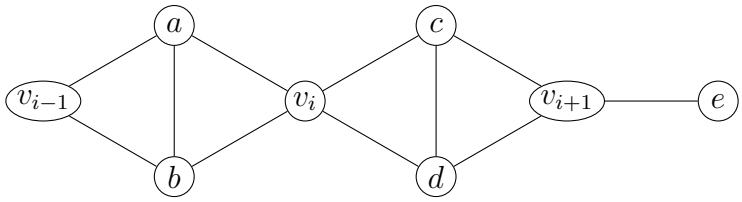


Figure 7.2: D_i, D_{i+1} and a pendant edge for Lemma 33

return distance 1 corresponding to response set $\{b, v_i\}$ and his second movement set is $\{v_{i-1}, b, v_i, c, d\}$. Finally probing v_{i-1} again forces him to return distance 3 corresponding to $\{c, d\}$, and his third movement set is $\{v_i, c, d, v_{i+1}\} = D_{i+1}$. Thus in three probes she moves him from D_i to D_{i+1}. Hence she can move him through the chain to v_k, or to v_0 by flipping these probes symmetrically, satisfying our first claim.

The second claim then follows immediately. The cop first uses the above technique to push the robber towards the single edge in the middle. Note that when pushed from one diamond to the next the robber occupies a set equivalent to $\{v_{i-1}, b, v_i, c, d\}$. But when this same set of probes is carried out pushing him right from D_{i+1} then he will occupy the set $\{v_i, d, v_{i+1}, e\}$, and then a probe at v_i will locate him. □

Lemma 34. The graph illustrated in Figure 7.1 is locatable.

Proof. We show this by outlining a simple strategy for the cop. With slight abuse of notation we refer to the chains of diamonds linking corner vertices as *threads*, since they will form similar roles to the actual path-like threads in other proofs. The cop first seeks a corner vertex which the robber is at most 10 away from by probing them in turn, stopping when she finds one. We can see that this shall succeed since he must begin on some thread, and since she probes the endpoints of that thread at most 3 steps apart he must be within distance 10 of one end when she probes it. Call this corner vertex v_i. She now seeks the other end of that thread by probing the other three corner vertices, alternating these with a probe back at v_i to prevent

him passing through v_i by the no back-track condition. As this takes at most 5 probes she will find the other end of the thread before he can leave it. She now knows that he is on a particular $v_i \cdots v_j$ thread, and has just probed v_j. Hence she knows exactly where he is on this thread, and hence by Lemma 33 can push him towards the middle edge and thus locate him. □

Thus we have shown that this graph is locatable. It remains to show that it is not 3-colourable.

Lemma 35. The graph illustrated in Figure 7.1 is not 3-colourable.

Proof. This is immediate by an application of the pigeonhole principle. Assume there is a valid 3-colouring of this graph. Then two of the corner vertices must receive the same colour. Consider the chains of diamonds linking these two vertices. They both start in the same colour, and hence the vertices at either end of the central edge must also be this colour. But then the central edge is monochromatic, contradicting our assumption that the 3-colouring was valid. □

We now consider what happens if we remove the no-backtrack condition. Firstly, note that without this condition Lemma 32 can be improved to show that any subgraph with minimum degree 3 is now a hideout graph, since the entirety of $N[x]$ shall be inside M_{t-1}. Lemma 31 also holds with 4 replaced by 3, or indeed any other positive integer. Hence combining these means that without the no-backtrack condition, every locatable graphs is 3-colourable. Showing that this is tight is then straightforward – to do so you just need to exhibit a graph that is locatable but not 2-colourable. Any sufficiently large odd cycle (at least 7 vertices suffices) will do for this. Hence without the no-backtrack condition similar results can be obtained, but they are arguably less interesting.

Part II

Sorting Algorithms

CHAPTER 1

INTRODUCTION TO SORTING ALGORITHMS

The work in this section is all joint work with Gábor Mészáros of Central European University. Our aim in this section is to study the following question. Assume a user has an ordered set of n elements in which the ordering is fixed but not known (for example distinguishable but unmarked coins of unknown distinct weights), and that he wishes to determine the ordering. We denote this base set X, containing elements x_1, $x_2 \dots x_n$. He is given a scale that accepts as input a k-set of elements and returns a fixed subset of them according to the ordering, for example it might return a subset of size s that contains the $t_1^{\text{st}}, t_2^{\text{nd}}, \dots, t_s^{\text{th}}$ elements. We would call such a scale a $(k, t_1, \dots, t_s)$ scale, and wish to know what one can determine about the ordering of the elements from repeated use of such a scale. We shall refer to the process of using the scale on a k-set as *querying* that set. These have been previously studied in the case where $k = 2$, in which case they are known as *binary* scales.

Clearly you cannot completely discover the ordering, as you cannot determine the ordering of the first $t_1 - 1$ elements or the final $(n - t_s)$ elements. Let us call these sets S for the initial segment and L for the final segment. Additionally if the scale is symmetric (in the sense that the elements that it returns are symmetric around the midpoint of k i.e. $t_1 = k - t_s, t_2 = k - t_{s-1}$ etc) then the ordering cannot be fully determined for the remaining elements, as the results of any query would be the same if the ordering was reflected. We therefore ignore this case, and assume assymmetric instruments in general. We also assume that the scale returns an unordered set, $\{t_1, \dots, t_s\}$, rather than an ordered one. This is because we show that even with an unordered set you can recover the full ordering of the elements; an ordered output would be strictly stronger, and so also able to do the same.

This section is structured as follows. In Chapter 2 we consider online

algorithms, where the sets submitted to future queries can depend on past results. Beginning with the case $s = 1$, i.e. where the scales output a single element, we show that it is possible to determine the ordering of the elements (excluding the ordering of S and L) in $O(n \log n)$ time, the same order of bound as in the binary case. We give this constant explicitly, showing that it is an improvement over the binary case. We also investigate the case where $s > 1$, and give an explicit algorithm for determining the ordering in this case.

In Section 3 we consider offline algorithms, where all the queries must be specified in advance and then the full set of results are returned simultaneously. This could be applicable in a situation wherein queries have to be sent to a laboratory to run overnight. Obviously this requires more queries in general. In the case where $s = 1$, i.e. where we are using a (k, t)-scale, we outline an algorithm that works in $O(n^{k-(t-1)})$ queries, and show that this is the best possible order. This algorithm relies heavily on a recursive approach, determining the results of queries that have not been carried out from those that have. We also outline an alternative algorithm that works in a similar amount of time, but works directly, determining the ordering of the elements using an adjacency based argument. In the case $s > 1$ the recursive approach can often still be applied, but the calculations involved get more complicated and require more detailed case analysis. However the adjacency argument continues to work, giving us a general offline algorithm that works when the scale outputs an unordered set.

CHAPTER 2

ONLINE ALGORITHMS

In this section we consider online algorithms, where the user is given the result of each query as he requests it, and on that basis selects the next set of k elements that he wants to query. As highlighted in the introduction, these scales cannot in general determine the full ordering of the element set, as the first and last segments of the ordering will never be returned by any query, and so no query can determine their order. We refer to these segments as S and L respectively, and in each case shall highlight which elements they comprise. The remaining elements we denote X', and note that there are at least $n-(k-1)$ of them, which we call n'.

We begin by considering a singleton-scale, which returns a singleton output.

2.1 Singleton Output Scales

The classical version of this question is the binary scale which accepts as input two elements and returns the smaller. We consider more general (k,t) scales, accepting k elements and returning the t^{th} smallest. To ease notation throughout this section we shall assume that $t \leqslant k/2$, i.e. it lies in the first half of the queried set. If $t > k/2$ then the following analysis still holds, inverting the roles of S and L and making occasional other minor adjustments to the calculations.

We describe an algorithm that works in $O\big(n' \log n'\big)$ queries to determine the ordering of the element set. This algorithm works by first determining S and L (in Stages 1 and 2), and then using them to iteratively determine the ordering by repeatedly determining the smallest element among the remaining unsorted elements (in Stage 3). Stage 3 shall take the longest, it is this stage that takes $O\big(n' \log n'\big)$ time to run, while Stage 1 is linear in n' and Stage 2 is independent of n', taking a number of queries that is just a function of k. Hence the combination of

the three works in $O\big(n' \log n'\big)$ queries as required.

Stage 1.1. Determine the elements that comprise $S \cup L$.

Method: Note that these elements are precisely those that shall not be returned by any query. Hence they can be identified by eliminating all the others. The user repeatedly picks a k-set from those that he has not yet eliminated, and eliminates whatever is the output. Each time he does this it eliminates 1 more element. He continues doing so until he cannot find another k-set, which occurs when there are $k-1$ elements left. But note that $S \cup L$ is always contained within the remaining elements, and there are $k-1$ elements in $S \cup L$, so it is exactly the remaining elements at this point.

Stage 1.2. Partition $S \cup L$ into S and L, and if possible identify which is which.

Method: Pick an arbitrary set from the eliminated elements, which we denote $a_1, \ldots, a_{k-1}$ where the index respects the ordering of the elements. For each element in $S \cup L$ query it taken together with $\{a_1, \ldots, a_{k-1}\}$. If it was in S this will return a_{t-1} and if it was in L it will return a_t. Although the user does not know which is which, he can partition $S \cup L$ into S and L according to which response he gets. He can further keep count of how often each comes up, as he will receive a_{t-1} $|S|$ times, and a_t $|L|$ times. If $|S| \neq |L|$ then this further identifies S and L – the only time this won't work is when $|S| = |L|$, which occurs when $t = k/2$, which means that the underlying scale is symmetric. Hence by the end of Stage 1.2 in the case of an asymmetric instrument the user knows S and L, and if the instrument is symmetric then he has identified the set $\{S, L\}$ but does not know which is which.

Stage 1.3. Use S to determine the order of the remaining elements.

Method: First, let us assume that our instrument is asymmetric, and hence the user knows S before starting this Stage – at the end we shall address what is done in the symmetric case. In this stage the user shall repeatedly use S to determine the

smallest element of a subset of $k' := k - (t-1)$ elements by querying them taken together with S. If he wishes to find the smallest element of a smaller set he takes it together with S and supplements it with elements from L until he has k elements allowing him to query it. In both cases as S takes up the first $t-1$ elements of the queried set, it will return the next largest which is the smallest of the subset he is trying to check.

The user now take the remaining n' elements and partitions them into as many sets of size k' as possible, with a remainder set of at most k' elements, which we shall call level 1 sets. He then groups the level 1 sets into sets of size k' (again as far as possible, with perhaps a deficient remainder set) to get level 2 sets, so a level 2 set consists of k' sets each of k' elements. He continues this process until all the n' elements are in a single set - we denote the level where this occurs as d, where $d = \log_{k'} n'$This effectively creates a d-dimensional grid of sets. We illustrate this grouping for the first two layers in Figure 2.1.

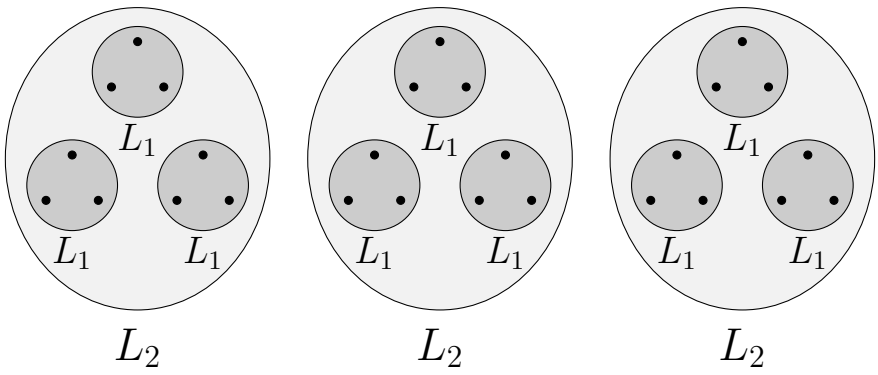


Figure 2.1: First two layers of the grouping process

At the first step he queries all the level 1 sets, establishing which is the smallest element of each. He then 'checks' each level 2 set in turn by querying the smallest element of each level 1 set inside it to find the smallest element in each level 2 set. Continuing in this manner he can establish the smallest element in every level r set for all $1 \leq r \leq d$, and hence the smallest element in the level d set. But the level d

set contains everything, so he has found the smallest element in the remaining n' elements. He now wants to remove this element and find the next smallest. Note that this only requires him to check/query the level $1, 2, \ldots, d$ sets that the previous smallest was in, as the others are unaffected. Repeating n' times therefore determines the order of the remaining elements.

This completes the analysis of the asymmetric case. If the instrument is symmetric, then recall as discussed earlier that the user would only at best be looking to determine the ordering or its reflection, since he would not be able to distinguish these with any query. If his instrument is symmetric though the main difference is that at the start of Stage 1.3 he wouldn't know which set was S and which was L. However he can arbitrarily assume either one is S, and carry out Stage 1.3 under that assumption. If he was correct he will get the correct ordering, if incorrect he will get the reflection, and given that he cannot distinguish these with a symmetric instrument anyway that therefore gives him the best possible information he could gather about the ordering.

It remains to show that this method only uses $O\big(n' \log(n')\big)$ queries. Stage 1.1 requires one query for each element in $X \setminus (S \cup L)$, so uses $n - (k-1)$ queries. Stage 1.2 requires one query for each element in $S \cup L$, so uses $(k-1)$ queries. Thus together Stages 1.1 and 1.2 use a total of n queries.

Stage 1.3 takes longer, this is where the extra log factor comes in. The first run through of the levels in which the user finds the smallest value in each set takes this many queries

$$\sum_{i=1}^{d} \left\lceil \frac{n'}{(k')^i} \right\rceil \leqslant dn'$$

Having completed these queries, and thus found the smallest element in the set $X \setminus (S \cup L)$, the user then needs to carry out d additional queries for each remaining element. Hence he requires at most $d(n'-1)$ remaining queries – in practice he may

require slightly less as when he gets some levels with only 1 element he doesn't need to continue to query them.

Hence in total this algorithm required at most the following number of queries

$$n + 2dn' = n + 2n' \log_{k'}(n') = O\big(n' \log n'\big).$$

2.2 Multiple output instruments

We now turn our attention to scales that return multiple elements. These accept as input k elements from X and return a set of size s containing the t_1^{st}, $t_2^{\text{nd}} \dots t_s^{\text{th}}$ elements. We refer to such a scale as a $(k, t_1, \dots, t_s)$ scale.

In this case we shall outline an algorithm that works in similar stages to the singleton output case, although in each case more work shall be needed to achieve the same ends. We shall again make use of the initial and final segments, which with slight abuse of notation we again denote by S and L. This time however S consists of the first $t_1 - 1$ elements, and L of the final $k - t_s$ elements. Together they therefore comprise a set of $k - 1 - (t_s - t_1)$ elements. It is possible if n is small compared to k that there can be other elements that the user cannot distinguish the order of. If, for example, we consider a $(7, 2, 6)$ instrument on 8 elements then none of $\{x_1, x_4, x_5, x_8\}$ will ever included in an output, so as well as the initial and final segments we have a middle segment that is indistinguishable. However in our case we think of n as being arbitrarily large compared to k; indeed we consider the asymptotics as $n \to \infty$, and if $n > 2k$ then no such middle set of indistinguishable elements can exist.

Stage 2.1. Determine the elements that comprise $S \cup L$.

Method: Again, as in the singleton case, $S \cup L$ comprise the elements that are never in the output of any query. So the user can begin by repeatedly querying uneliminated elements, and discarding the elements contained in the outputs. In the

singleton case, this worked until there were $k-1$ elements left, and these comprise exactly $S \cup L$. In this case however the user can only eliminate elements until he has $k-s$ left, and in general $S \cup L$ have $k-1-(t_s-t_1)$ elements. These are the same if $t_s - t_1 = s-1$, which occurs if the numbers t_1, $t_2 \ldots, t_s$ are consecutive. In that case Stage 2.1 terminates at this point.

If $t_1, \ldots, t_s$ are not consecutive then the user has a little more work to do. After carrying out the above he has $k-s$ candidates left for $S \cup L$, which contains $S \cup L$ and some extra elements which he wishes to eliminate. To do so we use an inductive approach. Specifically we shall show that if we have a set of $k-a$ candidates for $a < s$, $\{c_1, \ldots, c_{k-a}\}$ that contains at least 1 element not in $S \cup L$ then we can eliminate 1 more. This clearly suffices to eliminate all those not in $S \cup L$. Note that if we query $\{c_i\}$ along with a set of size a, as we get at least a outputs they must either consist exactly of the a additional elements, or include and thus eliminate an additional candidate. So to achieve the inductive aim, pick any set of $(2a-1)$ already eliminated elements, which we denote $\{e_1, \ldots, e_{2a-1}\}$, and carry out all the $\binom{2a-1}{a}$ queries involving a of the additional elements along with the candidates. Let x be an element in $\{c_i\}$ not in $S \cup L$, and note that by the pigeonhole principle either at least a of the $\{e_i\}$ are lower than x in the ordering, or at least a of them are greater – without loss of generality we assume the former, and relabelling if necessary we assume that the set $\{e_1, \ldots, e_a\}$ is among them. Hence when we query $\{c_1, \ldots, c_{k-a}, e_1, \ldots, e_a\}$, all of L and x are greater than all of the additional elements, hence the t_r^{th} element cannot be one of the additional elements and so must lie in the $\{c_i\}$, eliminating it. This process can continue until the remaining candidates are exactly $S \cup L$, after this point there does not exist any $x \in \{c_1, \ldots, c_{k-a}\} \setminus (S \cup L)$. Hence this process ends exactly with $S \cup L$ being identified.

Stage 2.2. Partition $S \cup L$ into S and L, and if possible identify which is which.

Method: We take exactly the same approach as in the singleton case. Pick any set of size $k-1$ taken from $X \setminus (S \cup L)$, which we denote $a_1, \ldots, a_{k-1}$. The user carries out the $|S \cup L|$ queries consisting of this reference set taken with one element from $S \cup L$. If the element from $S \cup L$ was in S then this returns $\{a_{t_1-1}, \ldots, a_{t_s-1}\}$, while if it was in L it returns $\{a_{t_1}, \ldots, a_{t_s}\}$. Although the user cannot immediately distinguish these they are clearly distinct, so he can partition $S \cup L$ into S and L according to the multiplicites of the responses.

In the singleton case the user could also determine which of these sets was S and which was L in the assymmetric case, since that implied they would be of different sizes. This is more complicated in the multiple-output situation, since you could have $t_1 + t_s = k + 1$, giving $|S| = |L|$, but still have an assymetric instrument as a result of some other elements in the output. In practice it doesn't matter which is which, and so for the moment we do not address the question of how to distinguish them in this algorithm. One of the referees kindly submitted a neat solution though, which we shall present separately at the end of this section.

Stage 2.3. Use S to determine the order of the remaining elements.

Method: If S and L are of different sizes, then the following works: Let S' be the set of the first $t_s - 1$ elements of X, noting that this includes S. If the user can identify S' then he can reduce our scale to a $(k', 1)$ scale by insisting on always including S' in any query – this fills up the first $t_s - 1$ slots of the scale, and means that it will always return some subset of S' (easily ignored) along with the smallest element of the remainder. Defining X' as $X \setminus S'$ he can then use this $(k', 1)$ scale to sort $X' \setminus L$ in $O\big(n \log n\big)$ steps as per Stage 1.3 of the singleton output algorithm. This sorts the majority of X (assuming as always that n is large compared to k. To sort the remaining elements in $S' \setminus S$ is then straightforward since the user will have identified the final k elements of X, so can create an (k'', k'') instrument by including $k - t_1$ elements of X. This can be used to sort the remaining elements.

It just remains therefore to say how to find S'. If $t_1 - 1$ divides $t_s - 1$ then S' is easy to find by repeatedly removing the smallest elements from X. In Stages 2.1 and 2.2 we outlined how to identify S i.e. the smallest $t_1 - 1$ elements in X. By removing these and repeating the process the user can identify the next $t_1 - 1$ elements repeatedly until he has found the first $t_s - 1$ elements. If $t_1 - 1$ does not divide $t_s - 1$ then the same process can be applied, except that when the user needs fewer than $t_1 - 1$ more elements at the end to top off S', he leaves in some of the previous $t_1 - 1$ set that he removed so that he requires something of the correct size.

If S and L are of the same size then at the end of Stage 2.2 the user has partitioned $S \cup L$ into sets A and B, which are S and L but he doesn't know which is which. We suggest the following: he makes an arbitrary assignment, claiming that A is S. Discarding this and repeating Stages 2.1 and 2.2 on the remaining $X \setminus A$ elements will return two more sets, A' and B. Note that this will be the same B as before, so he can identify A' and discard it again. Continuing in this manner he will eventually find a set that is either S' or L', as above. He can assume this is S', and use it to form what he thinks is an $(k', 1)$ instrument. Using this he can sort the remaining elements as above, and come up with an ordering for X. This ordering will either be correct, or exactly the reverse of the correct ordering if his initial theory that A was S was wrong. He can check which of these is true if he had an asymetric instrument by carrying out any query of k elements taken from $X \setminus (S \cup L)$, and seeing if the answer agrees with his opinion on what the ordering is. If it does not, he simply reverses the ordering.

If now remains to analyse how long this shall take. Stage 2.1 takes $\left\lceil \frac{n-(k-s)}{s} \right\rceil$ queries initially to get down to $k - s$ elements. To get from there to $S \cup L$ involves removing at most k elements, and removing each takes at most $\binom{2k-1}{k}$ queries, so in total this takes at most $(2k)^{k+1}$ queries. Stage 2.2 then takes $|S \cup L| = k - 1 - (t_s - t_1)$ queries. Hence between them Stages 2.1 and 2.2 take in

total a linear number of queries in n with an additional number of queries that is solely a function of k. As we consider the situation where n is large and k is small and fixed, this is effectively a linear number of queries in n.

Stage 2.3 takes as most k runs of Stages 2.1 and 2.2 to identify S', which is therefore still linear in n. Having identified S' it then takes $O\big(n \log n\big)$ steps to sort the set X'. The final sorting of the remanent, and the extra query in the case where S and L are the same size, clearly only take a number of steps that is a function of k, hence overall this algorithm also runs in time $O\big(n \log n\big)$ as required.

As promised, we now consider how to distinguish S from L in the case of an assymetric instrument in Stage 2.2. This is based on a suggestion from Prof. Paul Balister of Memphis. Let p be the smallest index that makes the instrument non-symmetric, in the sense that only one of the p^{th} and $(k+1-p)^{\text{th}}$ elements are in the output. Without loss of generality we shall assume it is the case that the p^{th} is in the output and its reflection (inside the scale) is not, to ease notation. We define $S_1 = S$, $L_1 = L$ and $X_1 = X \setminus (S \cup L)$, similarly to before, and then recursively define S_i, L_i and X_i to be the initial, final and middle segments of X_{i-1} respectively. Hence for any index i X is thus composed of $\bigcup_{j=1}^{i} S_j \cup \bigcup_{j=1}^{i} L_j \cup X_i$. Note that given that Stages 2.1 and 2.2 determined $\{S, L\}$ from X, they can be repeated to determine $\{S_i, L_i\}$ and X_i from X_{i-1}. Hence the user can build up as many pairs of sets as he wants, all of which form the initial and final segments of the remainder of the full set, provided n is sufficiently large.

We use this idea to show how to find S and L. First the user determines S_i and L_i for all $1 \leqslant i \leqslant p+k-2$. He can then identify S_p by querying a set containing one element from each of the sets in the pairs $\{S_1, L_1\}$, $\{S_2, L_2\}$, …, $\{S_{p-1}, L_{p-1}\}$, a single element from one of the sets in $\{S_p, L_p\}$ and then balancing elements from X_p to fill out the instrument. If the element he picked from $\{S_p, L_p\}$ was from S_p then it will be in the output, if it was from L_p then it won't, enabling him to identify S_p

and L_p. He can then repeat this to $\{S_{p+1}, L_{p+1}\}$, $\{S_{p+2}, L_{p+1}\}$, ..., $\{S_{p+k}, L_{p+k}\}$, and thus identify exactly the sets $S_p, S_{p+1}, \ldots, S_{p+k-2}$. Now identifying S is simple, since he just needs to take one element from one of $\{S, L\}$, and query it along with $k-1$ elements, taken one each from $S_p, S_{p+1}, \ldots, S_{p+k-2}$. The result of this query will determine if the element from $\{S, L\}$ was from S or L merely by looking at which of the other elements was returned, and hence he can identify which set is S and which is L. As p is at most $k/2$, if he wished to do this it would require at most $3k/2$ additional runs of Stages 2.1 and 2.2, which is still therefore linear in n and so would not materially affect the running time of the algorithm for large n.

CHAPTER 3

OFFLINE ALGORITHMS

We first turn our attention to offline algorithms. In this situation the user must specify the full list of queries that he wishes to carry out in advance, and then receives all the answers simultaneously afterwards. As we saw in Section 2, if the user knows the results of all the possible queries then he can determine essentially the full ordering (excluding S and L as before), since he can follow any of the online algorithms, looking up the results of any query from his bank of known query results. We concern ourselves with trying to minimise the number of queries that he must request in order to determine this ordering.

We begin by considering a singleton-scale, which returns a singleton output.

3.1 Singleton Output Scales

In this case, as in the online section on singleton output scales, the user is given a (k, t) scale that accepts a k-set as input and returns the t^{th} smallest element. Again for simplicity we assume that $t \leqslant k/2$, if not then similar analysis follows, occasionally replacing the word 'smallest' with 'largest' and, where we have comments about filling the scale from the lower elements, instead filling it from the largest.

We first note a trivial lower bound, namely there if there is any t-set that is not included in a query then there are some orderings that the algorithm would not be able to distinguish. This is because if the missed t-set comprised the first t elements of the ordering then no element of it would ever be included in the output of any set. Thus the user would not be able to tell which $t-1$ subset of it formed S, and which element was the lowest element of $X \setminus S$. The same applies if there was a $k-(t-1)$-tuple that was not included in any query, since it could form the largest

$k-(t-1)$ elements, and then the user would not be able to tell which was the largest element of $X \setminus L$. As $t \leqslant k/2$ by assumption, this second set is larger, and so there are more possible $k-(t-1)$ tuples than there are t-tuples. Hence this forms the restriction that we appeal to. Noting that there are $\binom{k}{k-(t-1)}$ $(k-(t-1))$-sets in any query, and as there are $\binom{n}{k-(t-1)}$ $k-(t-1)$-sets in total, this means that all algorithms must contain at least the following number of queries

$$\frac{\binom{n}{k-(t-1)}}{\binom{k}{k-(t-1)}} = O(n^{k-(t-1)})$$

We shall show that, in fact, there are algorithms that use this order of number of queries. We offer two for consideration, one that relies on recursively deducing the results of all possible queries, and thus the ordering, the second of which is direct and relies on determining the adjacencies of the ordering. We begin with the recursive algorithm.

3.1.1 Recursive Algorithm

Our algorithm works by fixing some set of r-elements, $Y := \{y_1, \ldots, y_r\}$, and requesting all the queries that involve Y and a $(k-r)$ set from $X \setminus Y$. We shall show that, provided r is not too large, then from this the user can deduce the result of an arbitrary query containing any $(r-1)$-subset of Y. If this holds then, by induction, the user can deduce the result of any query, and hence the full ordering. We prove this inductive claim by induction on r, beginning with the case $r = 1$.

Theorem 36. If y is a fixed element and the results of all queries including y are known, then the result of a query on any set $\{a_1, \ldots, a_k\}$ can be deduced.

Proof. Note that the claim is trivial if $y \in \{a_1, \ldots, a_k\}$, as this would mean that this exact query had taken place. So let us assume that $y \notin \{a_1, \ldots, a_k\}$. We wish to deduce the value of a_t from queries of the form $\{y, a_1, \ldots, a_k\} \setminus \{a_i\}$ for $1 \leq i \leq k$.

We shall split into 4 cases for y and 3 cases for a_i in relation to a_t, and count how often we get various responses. These are summarised in the following grid:

		Response			
	Multiplicity	$y < a_{t-1}$	$y \in (a_{t-1}, a_t)$	$y \in (a_t, a_{t+1})$	$y > a_{t+1}$
$a_i < a_t$	$t-1$	a_t	a_t	y	a_{t+1}
$a_i = a_t$	1	a_{t-1}	y	y	a_{t+1}
$a_i > a_t$	$k-t$	a_{t-1}	y	a_t	a_t

Now, when performing these queries we get all the results from some column. So if, for example $y < a_{t-1}$, then we get a_t $(t-1)$ times and a_{t-1} $(1+(k-t))$ times. We can establish which column by looking at the multiplicities of the answers - if we have two different answers with multiplicities $(t-1)$ and $(k-t+1)$ then we are in one of the first two columns, and if we get multiplicities t and $(k-t)$ then we are in one of the last two columns. Further if we get the answer y for some of our queries we are in the middle two columns, if not we are in the outside columns. Thus we can determine which column we are in. Now by taking the result with the appropriate multiplicity ($(t-1)$ in the first two columns, and $(k-t)$ in the latter two) we can tell the value of a_t, as required. We can summarise these in the following associated table:

Case	Multiplicities	Mult. of a_t
$y < a_t$	$(t-1, k-t+1)$	$t-1$
$y > a_t$	$(t, k-t)$	$k-t$

□

The general case is somewhat tricky to see, as the case analysis gets very detailed. Instead, we present the case for $r = 2$, which covers most of the concepts that we appeal to, and then explain how the argument changes for a general r. The

first key point is that just taking the queries involving x and y would not by itself be enough, as the user will also need to know which is larger out of x and y. But the following lemma gives a simple way to do that

Lemma 37. Assume that we have a asymmetric scale. Let $z_1, \ldots, z_{k+1}$ be $(k+1)$ fixed elements of our set. By querying all the $\binom{k+1}{k}$ subsets of them we can find two of them x and y such that neither x nor y are in $S \cup L$ and we know $x < y$.

Proof. Relabelling these reference elements according to the ordering, we note that any query of a subset of them will either return z_t or z_{t+1}. As these are possible responses to queries, neither can be a member of S or L so we take these as our x and y. It remains to show that the user can identify which is the smaller, but it is clear that the user will receive the answer z_t $(k+1-t)$ times, and z_{t+1} (t) times, enabling him to distinguish them if $t \neq \frac{k+1}{2}$. But this must hold, otherwise the scale would be asymmetric, which contradicts our assumption. □

We now need the next requirement, that given x and y, two fixed elements of the set such that the user knows that $x < y$, and all the queries involving this pair, the user can determine the results of any possible query, and hence as much of the ordering as could ever be possible.

Theorem 38. If x and y are two fixed elements such that $x < y$ and the results of all queries including x and y are known then the result of a query on any set $\{a_1, \ldots, a_k\}$ can be deduced.

Proof. Note that by Theorem 36 it suffices to show that we can find the result of any query involving x and an arbitrary set of other elements, $\{a_1, \ldots, a_{k-1}\}$. We shall proceed on this basis. Relabelling them we can refer to such a set as $\{b_1, \ldots, b_k\}$, noting that x is now one of the $\{b\}$, say b_i. We wish to establish how to find which is b_t from a set of queries in which we replace each of the b apart from b_i by y. The results for these queries – depending on whether the b_j that y replaces is

smaller than, equal to or larger than b_t – are summarised in the following table - note that now the multiplicities vary according to the position of $x = b_i$:

	Multiplicities			Responses			
	$x < b_t$	$x = b_t$	$x > b_t$	$y < b_{t-1}$	$y \in (b_{t-1}, b_t)$	$y \in (b_t, b_{t+1})$	$y > b_{t+1}$
$b_j < b_t$	$t-2$	$t-1$	$t-1$	b_t	b_t	y	b_{t+1}
$b_j = b_t$	1	0	1	b_{t-1}	y	y	b_{t+1}
$b_j > b_t$	$k-t$	$k-t$	$k-t-1$	b_{t-1}	y	b_t	b_t

We can again combine multiplicities according to $y < b_t$ and $y > b_t$. At first this looks like it won't let us differentiate options, as we get some situations with the same multiplicities:

Case		Multiplicities	Mult. of a_t
$y < b_t$	$x < b_t$	$(t-2, k-t+1)$	$t-2$
	$x = b_t$	$(t-1, k-t)$	$t-1$
	$x > b_t$	$(t-1, k-t)$	$t-1$
$y > b_t$	$x < b_t$	$(t-1, k-t)$	$k-t$
	$x = b_t$	$(t-1, k-t)$	$k-t$
	$x > b_t$	$(t, k-t-1)$	$k-t-1$

However we can see that this doesn't matter. Firstly the 2nd and 3rd rows of the above table are impossible, as our initial assumption was that $x < y$, so we can remove those. Secondly, we note that of the remaining 4 situations although 2 have the same multiplicities, in either of those two cases we just take the solution with multiplicity $k - t$ and conclude that this is b_t. Hence we can identify b_t for any arbitrary set of elements $\{b_1, \ldots, b_k\}$ such that x is one of them, and hence by Theorem 36 we can determine the order of the full set. □

We now give our method for the case $r = 2$. The user first picks a set of size $k+1$, and requests all the queries involving k of those – note that there are $(k+1)$ of these. By Lemma 37 this will find him a pair x and y from this set such that he knows $x < y$. He also considers all possible 2-tuples from this $k+1$ set, and for each pair requests all queries involving that pair. This means that, in particular, even though he is carries out these queries offline he will know the results of all possible queries involving x and y. In total this requires an additional $(k+1) + \binom{k+1}{2}\binom{n-2}{k-2}$ queries, which is of the same order as $\binom{n-2}{k-2}$, and hence $O(n^{k-2})$. But by Theorem 38 from these he can deduce the result of any query involving just one of x and y, and hence by Theorem 36 he can determine as much of the ordering as he could have hoped.

As advertised, we shall not give the full explicit argument for general r, as the analysis is tedious and not much more enlightening than the $r = 2$ case. We shall instead explain how to modify the $r = 2$ case. The first part is simple enough - carrying out all the possible probes on a $k+1$ set guaranteed us a pair of elements x, y such that we knew their internal ordering. In general, carrying out all the probes on some $(k+r-1)$ set guarantees us a set of r elements which we can completely order from these probes – this is simply seen by just taking those as our whole universe and using any argument such as that outlined in the online cases.

The notation for the other part gets more involved. We require the following statement by induction. Let $\{x_1, \ldots, x_r\}$ be our reference set which we know the complete ordering of. We want to be able to say that we can deduce the value of some query involving the first $(r-1)$ of the reference set and some set of elements $\{a_1, \ldots, a_{k-r+1}\}$ by considering all the queries containing the full r elements of the reference set and some $(k-r)$ subset of the a_i's. Let us relabel the set $\{x_1, \ldots, x_{r-1}, a_1, \ldots, a_{k-r+1}\}$ as $\{b_1, \ldots, b_k\}$, so that some of the b are taken from x-elements and some from a-elements. We shall then consider all the queries in

which we replace one of the a-elements by x_r. Note that when counting multiplicities we must consider where b_t lies relative to our reference set – i.e. how many of them are below it, and whether or not one of them is b_t. This gives rise to the following table of multiplicities, we have omitted the right-hand four columns as they are again the same as the above.

	# of reference set smaller than b_t				
	$(r-1) < b_t$	$(r-1) \leq b_t$	$(r-2) < b_t$	$(r-2) \leq b_t$	$\cdots$
$b_j < b_t$	$t-r$	$t-r+1$	$t-r+1$	$t-r+2$	$\cdots$
$b_j = b_t$	1	0	1	0	$\cdots$
$b_j > b_t$	$k-t$	$k-t$	$k-t-1$	$k-t-1$	$\cdots$

		# of reference set smaller than b_t			
	$\cdots$	$2 \leq b_t$	$1 < b_t$	$1 = b_t$	$0 < b_t$
$b_j < b_t$	$\cdots$	$t-2$	$t-2$	$t-1$	$t-1$
$b_j = b_t$	$\cdots$	0	1	0	1
$b_j > b_t$	$\cdots$	$k-t-r+3$	$k-t-r+2$	$k-t-r+2$	$k-t-r+1$

This then gives rise to the following table of multiplicities, where the lefthand column again corresponds to the number of reference elements below or equal to b_t.

Case		Multiplicities	Mult. of a_t
$x_r < b_t$	$(r-1) < b_t$	$(t-r, k-t+1)$	$t-r$
	$(r-1) \leq b_t$	$(t-r+1, k-t)$	$t-r+1$
	$(r-2) < b_t$	$(t-r+1, k-t)$	$t-r+1$
	$\vdots$	$\vdots$	$\vdots$
	$1 < b_t$	$(t-2, k-t-r+3)$	$t-1$
	$1 = b_t$	$(t-1, k-t-r+2)$	$t-1$
	$0 < b_t$	$(t-1, k-t-r+2)$	$t-1$
$x_r > b_t$	$(r-1) < b_t$	$(t-r+1, k-t)$	$k-t$
	$(r-1) \leq b_t$	$(t-r+1, k-t)$	$k-t$
	$(r-2) < b_t$	$(t-r+2, k-t-1)$	$k-t-1$
	$\vdots$	$\vdots$	$\vdots$
	$1 < b_t$	$(t-1, k-t-r+2)$	$k-t-r+2$
	$1 = b_t$	$(t-1, k-t-r+2)$	$k-t-r+2$
	$0 < b_t$	$(t, k-t-r+1)$	$k-t-r+1$

Again we can eliminate a large number of these situations. As we took x_r to be the maximum of the fixed reference elements, in the first half of this table all but the top row disappear, since if x_r is the largest and $x_r < b_t$ then all the other $(r-1)$ of them must also be less than b_t. In the second half each multiplicity is repeated twice, but we note that this doesn't affect our analysis as this is coming from separately counting the case where b_t is one of our reference elements and where it isn't. This however doesn't matter, as all we are interested in is the value of b_t, and either way we take the answer with the larger multiplicity (i.e. the second, as $t < \frac{k}{2}$), and so recover b_t.

This works all the way down to $r = t-1$. However when $r = t$ this no longer works, as you could be unlucky and pick as your fixed elements S together with the

smallest element left in the set. Then every probe would just give x_r as it will be the t^{th} element of any query. This is equivalent to noting in the above analysis that if all the first $(r-1)$ are less than b_t then if $x_r < b_t$ you'll end up trying to pick the element with multiplicity 0, which doesn't exist. Hence in this case you won't be able to identify b_t.

This matches the lower bound that we expect, so we conclude that this gives an construction requiring $\binom{k+t-2}{k} + \binom{k+t-2}{t-1} \cdot \binom{n-t+1}{k-t+1}$ steps to sort the set, $\binom{k+t-2}{k}$ to find a fixed reference set of size $(t-1)$ that we can fully order and then $\binom{n-t+1}{k-t+1}$ further steps to carry out all possible queries with each possible set of $(t-1)$ fixed elements. This is $O(n^{k-t+1})$ as required.

3.1.2 Adjacency algorithm

The key concept behind the second algorithm that we present is that of knowing which elements are next to which in the ordering. We begin with the following observation, which states that this would be sufficient to determine the full ordering of the element set.

Observation 39. Let $X := \{x_1, \ldots, x_n\}$ be a set of ordered elements, with the ordering being fixed but unknown to a user. Assume that the user knows which elements are adjacent to which others, i.e. he is given a map $\phi : X \to X^{(2)}$ such that

$$\phi(x_i) = \begin{cases} \{x_{i-1}, x_{i+1}\} & i \notin \{1, n\} \\ \{x_2\} & i = 1 \\ \{x_{n-1}\} & i = n \end{cases}$$

Then the user can deduce the ordering of the element set, up to reflection.

Proof. Note that only 2 elements only have 1 neighbour, namely x_1 and x_n. Hence the user can identify this pair easily. He picks one, and assumes it is x_1. He then proceeds iteratively – assume he has identified $x_1, \ldots, x_r$ up to some number

$1 \leqslant r < n$. He finds x_{r+1} by seeking elements who have x_r as a neighbour – there are at most two, at most one of which is x_{r-1} which he has already identified if it exists. He takes the other one as x_{r+1}, and thus extends the ordering. He can repeat this until he finally finds x_n, and ends up either with the correct ordering or, if his initial choice of x_1 was incorrect, the reflection of it. □

Hence it suffices to find a full list of the adjacencies to determine the ordering up to reflection. Note that, having done so if the instrument is asymmetric then any single query's result will determine which of the two possible orderings the user has, and if the instrument was symmetric then this would be the best he could have hoped for anyway. As ever, a full list will not be possible, but he can try to find all the adjacencies within $X \setminus (S \cup L)$. To do this the user will eliminate possible adjacencies for each element until only the actual adjacencies remain – and then appeal to the above observation to determine the ordering.

We suggest the following approach. Consider two elements from $X \setminus (S \cup L)$, x and y where x and y are adjacent. If any query returns the response x and then the same query is attempted with x replaced by y, then the second query must return the element y. Alternatively consider the situation where we have three elements, a, b and c, all taken from $X \setminus (S \cup L)$, with $a < b < c$, and as of yet the user does not know anything about their adjacencies. Consider a query of the form $\{x_1, \ldots, x_{k-2}\} \cup \{a, b\}$ which returns the output a. This means that a query of $\{x_1, \ldots, x_{k-2}\} \cup \{c, b\}$, with a replaced by c, cannot return a response of c since it would pick up b first. Hence if he carries out these two queries he will know that a cannot be adjacent to c - if it was by the first comment c would have had to be the response of the second query. However the existence of b between a and c ensures that the second query will not return c.

This motivates our approach – the user seeks a set of queries such that, for any triple (a, b, c) such that all three lie in $X \setminus (S \cup L)$ with $a < b < c$ he can find some

query containing a and b that returns a, and the same query with c replacing a. It suffices to fix some reference set of size $t-1$, which we call $y_1, \ldots, y_{t-1}$, and take all queries that contain these elements. If a, b, and c are all from $X \setminus (S \cup L)$, with $a < b < c$, then as at most $t-1$ of the fixed elements $\{y_i\}$ are less than a, the query that consists of these reference elements, a, b, enough elements from S to ensure that a is the t^{th} smallest and the remaining elements from L will return a.

Provided that a and c are not in this fixed reference set, as we include all possible queries containing the $y_1, \ldots, y_{t-1}$, the user will also see the result of the same query with a replaced by c, and will then be able to conclude that a is not adjacent to c. He will be able to do this for all the remaining c in $X \setminus (S \cup L)$, and thus be able to eliminate all of a's non-neighbours. He will be left with a's neighbours, and thus be able to deduce them.

If a or c are in the reference set, then this will not work. However we can circumvent this by carrying out three sets of queries, with disjoint fixed reference sets each time. Then, for any non-adjacent pair a and c, one of the three sets of queries must have neither a nor c in its reference set. Hence in at least one set of queries the fact that a and c are not adjacent will be revealed. Since the user will discover this for all the elements in $X \setminus (S \cup L)$ which are not adjacent to a, he will be left with those that are, and thus will be able to determine the ordering using Observation 39.

We note that this takes $3\binom{n-(t-1)}{k-(t-1)}$, which is again $O(n^{k-(t-1)})$ queries, but with a much improved constant factor over the previous recursive structure. It is also computationally less complicated, taking approximately n^2 calculations to eliminate all the non-adjacencies, and then a linear number of steps to rebuild the ordering, while the previous method required potentially reconstructing all the $\binom{n}{k}$ queries, a considerably larger task.

3.2 Multiple Output Scales

The question now arises of which of these algorithms also works in the Multiple-Output case, where the user is given a $(k, t_1, \ldots, t_s)$ scale as in the online analogue, and asked to determine the ordering. The authors note that the recursive algorithm was computationally and conceptually complicated even in the singleton output case – although we suspect it is possible to also use it for multiple output scales, the case analysis would make such an approach exceptionally tedious. However the adjacency based algorithm works almost immediately with almost no modifications. Since it just relies on showing that certain things would have to be included in the output if certain adjacencies existed, the same remains true even if the scale returns more elements. The only change required is that the fixed reference set is of a different size – before it contained at most t members, now it must contain at most the maximum of $t_s - 1$ and $k - t_1$ elements. This just ensures that the fixed reference elements don't take up so much of the scale that it's possible for every query containing them to only give an output consisting of members of the reference set. That established, the same analysis as before works, and so such an instrument can determine the ordering in at most the following number of queries

$$3 \max \left\{ \binom{n - (t_s - 1)}{k - (t_s - 1)}, \binom{n - (k - t_1)}{t_1} \right\}$$

REFERENCES

[1] M. Aigner and M. Fromme, A game of cops and robbers, *Discrete Appl. Math.* **8** (1984), 1–11.

[2] B. Bollobás, G. Kun and I. Leader, Cops and robbers in a random graph, *J. Comb. Theory, Ser. B* **103** (2013), 226 – 236.

[3] J. Carraher, I. Choi, M. Delcourt, L.H. Erickson and D.B. West, Locating a robber on a graph via distance scans, *Theoretical Comp. Sci.* **463** (2012), 54–61.

[4] N. E. Clarke and R. J. Nowakowski, Cops, robber, and photo radar, *Ars Combin.* **56** (2000), 97–103.

[5] P. Frankl, Cops and robbers in graphs with large girth and Cayley graphs, *Discrete Appl. Math.* **17** (1987), 301–305.

[6] J. Haslegrave, An evasion game on a graph, *Discrete Math.* **314** (2014), 1–5.

[7] F. Harary and R. A. Melter, On the metric dimension of a graph, *Ars Combin.* **2** (1976), 191–195.

[8] T. Łuczak and P. Prałat, Chasing robbers on random graphs: Zigzag theorem, *Random Structures & Algorithms* **37** (2010) 516 – 524.

[9] R. Nowakowski and P. Winkler, Vertex to vertex pursuit in a graph, *Discrete Math.* **43** (1983), 235–239.

[10] S. Seager, Locating a robber on a graph, *Discrete Math.* **312** (2012), 3265–3269.

[11] S. Seager, Locating a Backtracking Robber on a Tree, *Theoretical Computer Science* (2014), to appear

[12] P. J. Slater, Leaves of trees, *Proc. Sixth Southeastern Conf. Combin., Graph Theory, Computing* in Congressus Numer. **14** (1975), 549–559.