University of Glasgow | School of Computing Science

A Practical View of the Localisation Game

Jake Macaleb Haakanson (2407682H)

June 20, 2023

ABSTRACT

The localisation game is a pursuit-evasion game where an invisible target moves around the nodes of a graph and a probing player aims to locate it using distance queries. Many variations of the game have been studied giving bounds for several graph invariants. This project will explore the version of the game introduced by Carraher et al. [16] where a probing player with a single probe attempts to locate a target. Suzanne Seager developed a theoretical probing strategy for rooted trees on this version of the game [35], and analysing this strategy in practice is the primary focus of this project. A simplified version of Seager's strategy was developed throughout this project in order to assess several aspects of the strategy as there is no existing baseline to allow for comparisons. Sub-optimal target movement strategies were also developed in this project to enable testing of probing strategies in practice. Efforts were also made towards developing and implementing a MiniMax [39] model for the localisation game. While a MiniMax-based probing strategy seems infeasible without modifications to the model, the work carried out in this project shows promising findings relating to a MiniMax-based target movement strategy. Seager's strategy is also shown to be incredibly effective in capturing sub-optimal targets, exceeding theoretical bounds by a significant margin. Comparisons with the simplified version of the strategy show that the intricacy of Seager's strategy is necessary to ensure precise and efficient target capture.

1. INTRODUCTION AND PRELIMINARIES

Pursuit evasion games are a family of turn-based games where one group of agents attempts to *locate* or *capture* another. Graph-based pursuit evasion, or *graph searching*, was introduced by Parsons [29] in the 1970s, a graph defined as $G = (V, E)$ where V is a set of nodes or vertices and E is a set of unordered pairs of vertices known as edges. An edge between two vertices means that the two are adjacent and one can be reached from the other. The *localisation game* [34, 16, 26, 10, 8, 9] is a graph searching game played in rounds numbered increasingly starting from 1. The number of rounds will be represented as $r \geq 1$. A hidden target selects some node $t_1 \in V$ and in future rounds can move to any neighbouring node denoted t_r for the target's move on round r. On its turn, the probing player selects a set of any $k \in \mathbf{N}$ nodes in V and receives a vector of distances between these nodes and t_r. This is referred to as *probing*. Figure 1 shows a visualisation of an example round of a game. In a theoretical setting an optimal target with perfect knowledge is assumed and notation is defined as such. A graph G is *k-locatable* [26], or *locatable* [34] for $k = 1$, if the probing player is always able to theoretically capture the target in G with k probes in a finite number of rounds. The *localisation number* of a graph G, $\zeta(G)$, is the minimum k such that G is k-locatable [26]. The *location number* [34] of G, $loc(G)$, originally represented the number of rounds required to guarantee capture of the target in G, however this is now known as the *capture time* [1] of G.

> DEFINITION 1 (CAPTURE TIME). *$Capt_{\zeta,k}(G)$ for $k > \zeta(G)$ or $Capt_{\zeta}(G)$ for $k = \zeta(G)$ is the minimum number of rounds required for k probes to locate the target in G.*

These metrics are defined precisely in a theoretical setting where G is usually a general graph of some graph family adhering to certain constraints. As a practical investigation, this project is concerned with the notion of capture time in practice. This is further elaborated in section 5. Throughout the localisation game's history no investigation has explored practical implementations of the game. Probing strategies employing graph subdivisions [16] and multiple probes [26] have led to tight bounds on the localisation number and capture time for many graph families. Section 2 details the history of the game and its adaptations, including specifics of two theoretical strategies for rooted trees [35, 12].

The aim of this project is to conduct an experimental investigation on the effectiveness of Seager's strategy for rooted trees [35] in practice. Seager's work is foundational in introducing both the localisation game [34] itself, and the first general strategy for rooted trees [35], it therefore seems important to analyse the effectiveness of this strategy in practice. Section 3 outlines all probing and target player strategies used in this project. As there is no baseline for comparisons, a simplified version of Seager's strategy was developed in this project with a more relaxed approach to tracking the target's estimated location. This provides a utility for assessing Seager's strategy in two ways, by comparing the time taken to capture the target by both strategies, and determining if the intricate and precise nature of Seager's strategy is necessary to efficiently capture the target in terms of both run time and capture time. Details of probing strategies used in this project are found in section 3.1. Theoretical probing strategies are developed assuming an optimally playing target with perfect knowledge of all moves. This is necessary as it allows the definition of capture time and localisation number for many graph families. In a practical setting this is problematic as optimal target behaviour is not straightforward to implement, therefore sub-optimal target movement strategies have been explored, developed, and implemented. These are outlined in section 3.2. These implementations allow an analysis of the theoretical capture time bounds of Seager's strategy in practice. Seager's strategy and the simplified version are compared in section 5, allowing an assessment of both run time and capture time. The two metrics are related in a practical setting as larger capture time leads to longer run time, however this leads to a discussion of algorithm efficiency. Lastly, developments towards implementing a MiniMax [39] model for the localisation game are discussed in section 3.3, the main aim of exploring this model is to investigate the possibility of optimal target movement in practice. Section 6 summarises the project's findings, discussing limitations and challenges, and outlining opportunities for further research.

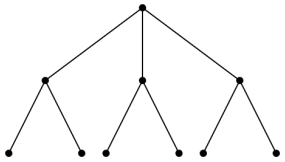
1.1 Scope of the Project

Seager's strategy [35] is given for general rooted trees, therefore the project will explore this graph family. Future references to trees throughout this report refer to rooted trees.

> DEFINITION 2 (ROOTED TREE). *A rooted tree [2] is a tree with a designated root node m on level 0. Every node $u \neq m$ shares an edge with a parent node v forming a path back to the root. u is a child of v. Each node is located on a level in the tree, where a node on level i has a parent on level $i-1$ and children on level $i+1$. For nodes on the same path to the root, nodes on upper levels are referred to as ancestors and on lower levels as descendants.*

The version of the game used in this project restricts k to 1 meaning that the probing player can probe 1 node per round. The target can also move to the node probed in the previous round or *backtrack* [16]. Seager's strategy, the primary focus of this project, was developed on this version of the game, introduced by Carraher et al. [16]. A *hideout* is a subtree that allows the target to evade capture indefinitely. In practice a sub-optimal target may not be able to evade capture indefinitely, however since this project concerns the analysis of theoretical strategies which function on *hideout-free* rooted trees, trees used in this project will not contain hideouts.

> DEFINITION 3 (HIDEOUT). *A hideout consists of a node with degree ≥ 3 with at least 3 neighbours of degree ≥ 3. For $k = 1$ the target can evade capture indefinitely [35].*



Example hideout for $k = 1$, adapated from Seager [35].

2. BACKGROUND

This section details the history of the localisation game, the variants developed with updated rules for the players, and existing theoretical strategies relevant to this project. Section 2.1 discusses early graph theory developments that play a significant role in the mathematical background of the game, and section 2.2 covers the formal introduction of the game. Section 2.3 details a variant of the game with updated rules and section 2.4 introduces Seager's strategy [35] on this version of the game. Another probing strategy based on Seager's original game [34] is briefly introduced in section 2.5. Section 2.6 discusses more modern developments, such as capture time [1], and other variants of the game, such as the localisation game with more than one probe [26]. Finally, section 2.7 introduces the Mini-Max decision rule [39].

2.1 Early Ideas

Ideas motivating the localisation game date back to the 1970s [38, 24] and involve finding a *resolving set* [14] or the *metric dimension* of a graph. Given a graph $G = (V, E)$ a resolving set of G, $S \subset V$, is defined such that the vector of distances between any node in V and the nodes in S is unique. The metric dimension of G is the size of a minimum size resolving set and determining these sets is known to be NP-hard in general graphs [22]. The metric dimension ties in closely with the localisation game [4] as it effectively gives an upper bound on k in that $k \geq |S|$ nodes would locate the target in a single round. In the case of $k < |S|$ then S provides the set of nodes in G that should be probed to most effectively locate the target. Some versions of the localisation game can be viewed as a game-theoretic approach to determining a resolving set for a graph [4]. A sequential approach to determining a resolving set has been studied [14] where a set is gradually built by marking individual nodes as belonging to S until a number of key nodes in V are identifiable. This same idea has been employed in a sequential version of the localisation game [36, 3] where the key node is occupied by an immobile target. Each probe provides distance information informing the next and the goal is to locate the target in as few probes as possible. It should be noted that the rules of the sequential version differ greatly from more modern versions of the localisation game and therefore the ideas are not entirely relevant to this project.

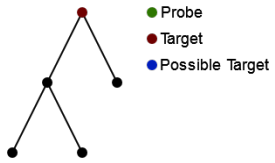
2.2 The Game's Formal Definition

The localisation game was formally introduced by Seager for a mobile target and single probe each round [34]. Seager's version features a rule later referred to as the *no-backtrack condition* [16], meaning that the target is not allowed to move to the node probed in the previous round. Seager proved many properties on this version of the game, such as that any path P has $Capt_1(P) = 1$ and cycles with $|V| \neq 5$ are locatable. Seager also showed that any tree T with $|V| \geq 3$ has $Capt_1(T) \leq |V| - 2$ with the *root location property*, a rule where the target is located if it ever moves to the root of the tree. Seager's work is foundational as it introduces the game and many properties used in future work, however this version of the game with the no-backtrack condition and root location property, which greatly increase the strength of the probe, is only used in one strategy paper [12]. Carraher et al. later showed that, due to this increased strength, many properties also hold with backtracking [16]. This resulted in Seager updating theorems for trees and developing an effective strategy [35] for this version of the game.

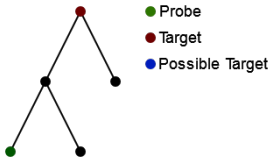
2.3 Backtracking and Graph Subdivision

As briefly mentioned, Carraher et al. [16] expanded Seager's original work to include backtracking. They showed that many properties still hold, for example cycles with $|V| \geq 7$ are locatable. Their primary focus was the subdivision of graphs. Given a graph G the subdivision of G, denoted $G^{1/m}$, is obtained by replacing each edge of G with a path of length m. Carraher et al. showed that any subdivided graph $G^{1/m}$ is locatable when $m = 3 \cdot |V|$, a subdivided complete bipartite graph $K_{a,b}^{1/m}$ has a capture time of $b + m \cdot (b + a)$, a complete graph $K_n^{1/m}$ is locatable when $n \geq 4$ and $m \geq n$, and a grid $G_{k,l}^{1/m}$ has a capture time of 3 when $m \geq 2$. They also conjectured that these bounds are best possible, however Seager [35] and Haslegrave et al. [12] later independently showed that graphs of girth[1] 6 are locatable and that there is a strategy for locating the target in the subdivision of a four vertex complete graph $K_4^{1/3}$, a counterexample to the earlier claim that m must be greater than or equal to n. Haslegrave et al. improved the bounds to the best possible, showing that both $G^{1/m}$ and $K_n^{1/m}$ are both locatable when $m \geq \frac{|V|}{2}$ [25]. Carraher et al. greatly contributed to the localisation game by updating many previously known graph properties to allow for backtracking. Their work also demonstrates that graph transformations can make a graph locatable, however these transformations are not property preserving as the increased path length between original nodes causes the target to move more

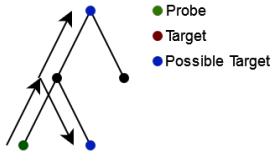
[1]The girth of a graph is the length of the shortest cycle in the graph.



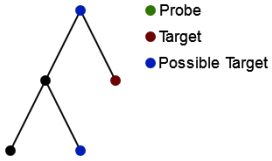
1. The target initially selects some node of the tree.



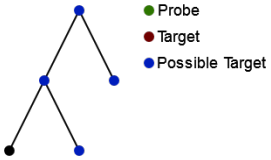
2. The probing player then selects some node of the tree ($k = 1$).



3. A distance of 2 is returned, showing two possible locations.



4. The target moves as it has not been located.



5. The target set is expanded to account for this move.

Figure 1: Visualisation of the steps in one round of a game.

slowly between two original nodes. This means that a subdivided graph being locatable does not imply the same for the original.

2.4 Seager's Localisation Strategy for Trees

Seager created a strategy for trees and characterised trees that are not locatable [35] on the version of the game created by Carraher et al. [16]. Seager introduced the concept of a hideout, as in definition 3, and proved that the target can evade capture if such a subtree exists[2]. Given a rooted tree, Seager's strategy first organises the tree such that each node's highest degree child is its leftmost child, and its second highest degree child its rightmost. With this ordering it is possible to detect hideouts[3] and determine an appropriate course of action given the distance returned by each probe[4], with the initial probe being the root of the tree. Seager discusses the strategy's steps showing a relatively poor worst case capture time of $5 \cdot n$ for an n node tree, however as observed in section 5 the approach performs much better in practice. Seager's work on trees is highly theoretical and from an implementation perspective the preprocessing required to organise the tree and search for hideouts could be computationally inefficient. The characterisations given by Seager are very useful, as not only do they show that a winner can be determined during preprocessing, but also direct the probing player to follow certain sets of moves depending on tree structure.

2.5 Another Localisation Strategy for Trees

Only one strategy makes use of Seager's original version of the game [12]. Given by Brandt et al., the strategy is also specific to trees. In this strategy probes are used to eliminate certain tree components in order to narrow down the target's location. The strategy relies on the root location property, a feature not employed in any other strategy due to the greatly increased probe strength. The use of this coupled with the no-backtrack condition mean that this strategy is not applicable to this project. The strategy is motivated by Seager's proof that $Capt_1(T) \leq |V| - 2$ for any tree $T = (V, E)$ and shows that this theoretical bound can be met and improved by an acceptable margin, promising results improving on earlier bounds.

[2] See theorem 7 of [35].

[3] See lemma 1 of section 3.1.1 [35].

[4] See theorem 8 [35]

2.6 The k-Probe Version

2.6.1 *Introduction of the k-Probe Version*

With the favourable results of graph subdivision an extension to the game was introduced by Haslegrave et al. where the probing player places a set of k probes per round [26]. Two graph properties were introduced, $rl_p(G)$, the smallest k for which G is k-locatable, now denoted $\zeta(G)$, and $rl_s(G)$, the smallest m for which the subdivided graph $G^{1/m}$ is locatable. The subscripts p and s represent probes and subdivisions. Haslegrave et al. explored these properties in great detail both for general graphs and for graphs with bounded degree, the relationship between the two was also explored and it was shown that $rl_s(G) \leq (2 + o(1))rl_p(G)$. Haslegrave et al. also proved that the subdivided graph $G^{1/m}$ is locatable when G is k-locatable and $m \geq 2 \cdot k + 2$. The k-probe version marks the beginning of more modern extensions to the localisation game. Many properties were proven in relation to whether or not graphs are k-locatable, and this is tied in to the subdivision of these graphs. The extension also added to existing notation with the localisation number $\zeta(G)$ where earlier versions of the game restrict k to 1.

2.6.2 *k Probes and Blind Probes*

Bosek et al. studied this version in the context of cellular networks [10], their main contribution being several upper bounds for the value of $\zeta(G)$ in general graphs and many graph families such as trees and bipartite graphs. They showed that the localisation number of all connected bipartite graphs $G_{a,b}$ is less than or equal to $min(a, b)$ and that the localisation number of all graphs are bound by their pathwidth. The pathwidth of a graph is the minimum width of a path decomposition of the graph, a path decomposition being the splitting of the graph into subsets with sequentially ordered nodes and edges [32]. Bosek et al. briefly introduced another ver-

sion of the localisation game with *blind probes*, probes which can only view the neighbourhood of a node. Blind probes were later studied by Bonato et al. who formalised the version of the game as the *one-visibility localisation game* with the *one-visibility localisation number* denoted ζ_1(G) [8]. One-visibility probes have been studied for several graph classes, however this strays far from the version of the game used in this project. Bosek et al. contributed to progressing the localisation game and showed some interesting results such as that the localisation number of a graph is bound by its treewidth and that the game itself, determining a minimum $\zeta(G)$ in general graphs, and determining $\zeta(G)$ in graphs with diameter 2 are all NP-hard.

2.6.3 Centroidal Localisation

Later work by Bosek et al. introduced the *centroidal localisation game* and the *centroidal localisation number* denoted $\zeta^*(G)$ [9]. In this variant the probing player selects a set of k nodes, $\{v_1, ..., v_k\}$, in each round and is only able to compare whether the distances returned by successive probes are greater than, equal to, or less than each other. For example for target at node t, a set of probed nodes $\{v_1, ..., v_k\}$, and $1 \leq i < j \leq k$ the probing player can determine if $d(v_i, t) = 0$, $d(v_i, t) = d(v_j, t)$, $d(v_i, t) < d(v_j, t)$, or $d(v_i, t) = d(v_j, t)$ where the function $d(u, v)$ returns the distance between two nodes u, v. This variant is motivated by the simulation of detecting a mobile device on a wireless network where the comparative strengths sequential multiple probes is considered more accurate for detection of the target in the real world. The centroidal localisation number of a graph G, $\zeta^*(G)$, is the minimum k for which the probing player is able to win on G regardless of target strategy. On this version of the game Bosek et al. proved that $\zeta^*(T) \leq 2$ for any tree T, $\zeta^*(G) \leq 3$ for any outerplanar graph G. The authors also showed that $\zeta^*(G)$ is bounded by the pathwidth of G plus 1 and finally that the task of determining $\zeta^*(G)$ in general graphs is NP-hard. This work is significant in showing more of a practical application of localisation game variants, and while this project does not cover the centroidal localisation game, practical application is something that this project does aim to address. In this context the metric dimension is referred to as the centroidal dimension [21], and the variant remains a game theoretic approach to determining this set.

2.6.4 Cartesian Products

The cartesian product of graphs is denoted as $L = G \square H$ for two graphs $G = (V_G, E_G)$ and $H = (V_H, E_H)$ [33]. The vertex set of L is defined as the cartesian product $V_G \times V_H$ with vertices represented as pairs of vertices from each of the original graphs. Given two vertices in L, (u_G, u_H) and (v_G, v_H), the vertices share an edge if and only if either $u_G = v_G$ and u_H is adjacent to v_H in H, or if $u_H = v_H$ and u_G is adjacent to v_G in G. Earlier work on the centroidal localisation game by Bosek et al. [9] briefly investigated the centroidal localisation number on cartesian products, giving an upper bound of $\zeta^*(G \square H) \leq \max\{(\Delta(G)+\Delta(H)+1), (\Delta(G)+\zeta^*(H)), (\zeta^*(G)+\Delta(H))\}$ where the delta function gives the maximum degree of a graph. Boshoff and Roux [11] further studied the localisation game on cartesian products where the more general upper and lower bounds of $\zeta(G \square H) \geq \max\{\zeta(G), \zeta(H)\}$ and $\zeta(G \square H) \leq \zeta(G) + \psi(H) - 1$ were given. Here $\psi(H)$ denotes the doubly resolving number of H [15]. The doubly resolving number of a graph is the minimum cardinality doubly resolving set, and a doubly resolving set W of a graph G is a set such that every distinct pair of vertices $u, v \in G$ are identifiable by another two vertices in W.

2.6.5 Further k-Probe Developments

Several other approximate and probabilistic bounds were given by Dudek et al. [20] under the k-probe version. They showed that $\zeta(G) \approx 2 \cdot \log_2(|V|)$ for most graph families and several other important properties were proven for general graphs. A conjecture made by Bosek et al. in the context of cellular networks [10] was resolved by Bonato and Kinnersley where they gave the chromatic number of a graph as a function of the localisation number [7], primarily showing that every G with $\zeta(G) \leq k$ has degeneracy less than 3^k. The degeneracy of G is the smallest value of k for which G is k-degenerate. A k-degenerate graph can be decomposed such that every subgraph contains a node of degree k. Bonato and Kinnersley also showed that $\zeta(G) \leq 2$ for any outerplanar graph. An outerplanar graph can be embedded in the plane as a shape where all vertices belong to the outer face of the shape. Bosek et al. earlier proved that it is NP-hard to determine $\zeta(G)$ in graphs of diameter 2 [10], and this was further studied particularly for Kneser and Moore graphs, and graphs that do not contain 4-cycles by Bonato et al. [4, 6]. In this work, the value of $\zeta(G)$ was closely tied in with the metric dimension of these graphs where upper and lower bounds on the localisation number were given. Bounds were later given for the localisation number in incidence graphs by Bonato et al. [5, 6], random geometric graphs by Lichev et al. [27], and bounds were improved for random dense graphs by Dudek et al. [19]. This summary of the k-probe version outlines the current direction of the topic with recent work studying bounds on the localisation number for many graph families.

2.6.6 Capture Time

More recent analysis of the k-probe version by Behague et al. introduced the capture time of graphs [1], as in definition 1. Capture time was originally introduced by Seager as $loc(G)$ [34]. The work of Behague et al. formally separated the capture time from the location number in settings where $k \geq \zeta(G)$. The authors also introduced the notion of a *well-localisable* graph. A graph is *well-localisable* when the capture time is at most linear in the size of the graph and it was proven that trees and certain other graph families are well-localisable. Behague et al. also showed that the capture time is monotone for induced subgraphs in trees and some general graph families, this means that the capture time of the subgraph is less than or equal to the capture time of the graph. Capture times were also given for other graph families, such as $Capt_{\zeta,j}(T) = 1$ for a tree T with j leaves. This work serves to progress the game in providing an explicit and updated definition of the location number for k probes, solidifying modern notation.

2.7 MiniMax

MiniMax is a decision rule in game theory where a negative player aims to minimise its score and a positive player aims to maximise its score. The MiniMax theorem was originally introduced in 1928 [39] and is largely seen as the start of game theory. The theorem has been adapted in recent years for various purposes [18, 13, 37, 30] which aim to address issues with the original that arise in more specific contexts. Many subsequent improvements have addressed issues resulting in a widely applicable approach to many one- and two-player, and zero-sum games. The duality theorems for dual linear programming were introduced to prove that MiniMax is applicable to zero-sum games [28].

3. STRATEGIES

This section provides an overview of the various probe and target strategies used in this project, implementation specific details are

given in section 4. A description of probing strategies is given in section 3.1 followed by a discussion of target strategies in section 3.2. The following notation, originally given by Seager [35], is used throughout this section. A tree is a fully connected graph defined as $T = (V, E)$ with no cycles, where $n = |V|$ and $t \in V$ represents the node currently occupied by the target. The game is played on rooted trees as in definition 2. For two nodes, u, v, on the same level in the tree, $u \leq v$ means that u is to the left of or the same vertex as v as the nodes are ordered. The set $\{u, ..., v\}$ represents all consecutive nodes between and including u and v with $u \leq v$. *Children*(u, v), *Children*$(\{u, ..., v\})$, or *Children*(v) when $u = v$, is the combined set of all children of nodes in $\{u, ..., v\}$ and *Parents*(u, v) is similarly defined to retrieve the combined set of all parents. *Siblings*$(u, v) = \{u, ..., v\}$ when u and v have the same parent. $succ(v)$ represents the node ordered directly after v on the same level, and $pred(v)$ represents the node ordered directly before v. The *target set* R is a set of estimated target locations determinable after a certain set of probes and the function $probe(v)$ returns distance d between v and t. The target set is divided into three subsets, D_{j-1}, D_j, D_{j+1}, where j is level in the tree and nodes in the subsets fall on the respective level. Trees are arranged such that each node's leftmost child is its highest degree child and its rightmost child is its second highest degree child. This enables strategies to function correctly and also enables the detection of hideouts, defined in definition 3. As a tree is a theoretical object with no particular ordering this rearrangement does not affect the underlying structure of the tree. In the event that a tree must adhere to a particular ordering then certain strategies, such as Seager's, will not function correctly on the tree.

3.1 Probing Strategies

3.1.1 Seager's Strategy

Seager described the first implementable strategy for trees in [35]. The strategy centres around marking a level in the tree as level j[5]. This is the level in the tree that t is known to occupy after a certain set of probes and is used to maintain R. R can be determined after the initial probe, the root, as D_j is simply level j of the tree, which is the distance returned by the probe. D_{j-1} and D_{j+1} are the parents and children of each of the nodes in D_j respectively, and v_j is the leftmost node of D_j. Lemma 4 of the strategy acts as a controlling function which probes the leftmost child of v_j and directs the probing player to follow a certain set of subsequent probes to either locate t or find nodes u', v' such that $t \in$ *Children*(u', v'). The level of *Children*(u', v') becomes level j when returning to lemma 4, and with this return to lemma 4 the sets can be re-established. Seager gives the strategy as four lemmas [16], a brief outline of each is given below.

Lemma 1: Lemma 1 shows that given some node v, only the first and last children of the node can have more than one child due to the way the tree is arranged. The only exception is when v's parent has more than one child, in which case only v's leftmost child can have more than one child. If this criteria is not met then a hideout exists in the tree and the target will always win with perfect play.

Lemma 2: Given two nodes w, z with the same parent v and $w < z$, lemma 2 handles the scenario where $t \in$ *Siblings*(w, z) and one of w or z has at most one child, for example w. This means that all nodes $\{w, pred(z)\}$ will have at most one child. The lemma functions recursively $succ(w), z$ where $w = succ(w)$ in the next execution until t is either located or is in a subset of *Children*(w, z). The lemma probes w's child or the node itself if it has no children and either determines t's location, recursively returns lemma 2 on $succ(w), z$, or returns to lemma 4 where t is in a subset of *Children*(w, z). A similar strategy is employed for z with at most one child where the lemma is returned to recursively with $w, pred(z)$.

Lemma 3: Lemma 3 is a special case of lemma 2 where v, with children w, z, is the unique child of its parent u. In this case u is probed initially and lemma 2 is called for $t \in$ *Siblings*(w, z), or lemma 4 for $t \in$ *Children*(w, z).

Lemma 4: Lemma 4 is the controller of the strategy and handles the case where, given two nodes u, v, the $t \in$ *Children*(u, v), determined initially after the root of the tree has been probed or after a set of cases and lemmas have been followed. Each time lemma 4 is given nodes u, v, level j is updated to the level of *Children*(u, v). v_j is assigned to be the leftmost child of u. Lemma 4 probes the leftmost child of v_j, v_{j+1}, if it exists or v_j if it does not, and directs the probing player to the necessary case based on the distance d_1 returned by this probe. Each case is briefly described below.

Case 1: For $d_1 = 2$, $D_j = \emptyset$ and $t \in R = \{u\} \cup D_{j+1}$. If v_j is the unique child of u then lemma 3 follows, otherwise given the rightmost child of u, z_k, $d_2 = probe(z_k)$ is obtained. If d_2 is 2, 3, or 4, then lemma 3, lemma 2, or lemma 4 follow respectively.

Case 2: For $d_1 = 3$, $t \in R = D_j =$ *Chilren*$(u) - \{v_j\}$ and lemma 2 follows.

Case 3: For all odd $d_1 > 3$, $t \in R =$ *Children*(w, x) where $w = Succ(u)$ and x is the rightmost child of the rightmost node of D_{j-1}, as a result lemma 4 follows.

Case 4: For $d_1 = 4$, $D_j = \emptyset$ and $t \in R = D_{j+1} \cup D_{j-1}$ where D_{j-1} are siblings. If $D_{j+1} = \emptyset$ then lemma 2 follows on D_{j-1}. If $D_{j-1} = \emptyset$ then lemma 4 follows on D_{j+1}. z_{k-1} is the rightmost node of D_{j-1} s is its leftmost child and z_k is its rightmost child, $d_2 = probe(s)$ if s exists otherwise $d_2 = probe(z_{k-1})$ and distances are adjusted accordingly. If $d_2 < 2$ then t is located. If $d_2 = 3$ then t is in *Children*$(z_{k-2}) - \{u\}$ which are siblings, therefore lemma 2 follows. If $d_2 = 4$ then t is in the children of *Chilren*$(z_{k-2}) - \{z_{k-1}\}$ which are not siblings and lemma 4 follows. For $5 < d_2 < 6$, t is in a set of descendent nodes to the case where $d_2 = 4$ and therefore lemma 4 follows on the determined sets of nodes. If $d_2 = 2$ then $t \in \{z_{k-2}\} \cup ($ *Children*$(z_{k-1}) - z_k)$ and $d_3 = probe(u)$. If $d_3 = 2$ then $t \in R = \{z_{k-3}\} \cup ($ *Children*$(z_{k-2}) - u)$ and lemma 3 follows. If $d_3 = 3$ then $t \in$ *Siblings*$(succ(s), z_k)$ and lemma 2 follows. Finally, if $d_3 = 4$ then $t \in$ *Children*(s, z_k) and lemma 4 follows.

Case 5: For all even $d_1 > 4$ then $D_j = \emptyset$ and $t \in R = D_{j-1} \cup D_{j+1}$. If $D_{j-1} = \emptyset$ or $D_{j+1} = \emptyset$ then lemma 4 follows on the non-empty set respectively, otherwise with z_{k-1} as the rightmost node of D_{j-1} one of three subcases follows.

Case 5a: z_{k-1} is the rightmost child of z_{k-2}. $d_2 = probe(v_\ell)$ where v_ℓ is the ancestor of v_{j-1} on level $\ell = \frac{d_1}{2}$. Lemma 4 follows for $d_2 = \frac{d_1}{2} + 2$, $d_2 = \frac{d_1}{2} + 1$, $d_2 = \frac{d_1}{2}$, or $d_2 = \frac{d_1}{2} - 1$. If $d_2 = \frac{d_1}{2} + 2$ then $t \in$ *Children*(D_{j+1}). If $d_2 = \frac{d_1}{2} + 1$ then

[5]This is level k in Seager's original paper, however this has been renamed to avoid collision with the k-probe variant.

$t \in D_{j+1}$, If $d_2 = \frac{d_1}{2}$ then $t \in$ *Parents*$(D_{j+1}) \cup$ *Children*(D_{j-1}). If $d_2 = \frac{d_1}{2} - 1$ then $t \in D_{j-1}$. Finally there are two possibilities if $d_2 = \frac{d_1}{2} - 2$. If $d_2 = \frac{d_1}{2} - 2 \neq 6$ then $t \in$ *Parents*(D_{j-1}) and lemma 4 follows. If $d_2 = \frac{d_1}{2} - 2 = 6$ then $v_{k-3} = z_{k-3}$ meaning *Parents*(D_{j-1}) are siblings, therefore lemma 2 follows.

Case 5b: z_{k-1} is the first of many children of z_{k-2} and z_{k-1} has more than one child. $d_2 = probe(z_{k-2})$. If $d_2 < 2$ then t is located, and if $d_2 = 2$ then $t \in$ *Children*(z_{k-1}) and lemma 2 follows, otherwise lemma 4 follows for each of the following distances. If $d_2 = d_1$ then $t \in$ *Children*(D_{j+1}), if $d_2 = d_1 - 1$ then $t \in D_{j+1}$, finally if $d_2 = d_1 - 2$ then $t \in$ *Parents*(D_{j+1}). Seager describes a special case that is not outlined here [35].

Case 5c: z_{k-1} is not the last child of z_{k-2} and z_{k-1} has at most one child. $d_2 = probe(z_k)$. If $d_2 < 3$ then t is located. If $d_2 = 3$ then $t \in$ *Siblings*$(q, pred(z_{k-1}))$ where q is z_{k-2}'s leftmost child and lemma 2 follows. If $d_2 > 4$ then the target set is the same as for subcase 5b, so subcase 5b follows. If $d_2 = 4$ then $d_3 = probe(z_{k-3})$. If $d_3 = 1$ then $t \in$ (*Children*$(z_{k-3}) - z_{k-2}$) and lemma 2 follows. If $d_3 = 2$ then the target set is a subset of $R - z_{k-1}$, therefore case 5 is repeated with $pred(z_{k-1})$ replacing z_{k-1}. This repetition can only occur while the two are siblings. Finally if $d_3 = 3$ or $d_3 = 4$ then lemma 4 follows as either $t \in$ *Children*$(q, pred(z_{k-1}))$ where q is the leftmost child of z_{k-2} or t is in the children of the set for $d_3 = 3$.

3.1.2 Simplified Seager

The development of the simplified Seager strategy throughout this project was motivated by the goal of reducing the intricacy of Seager's strategy and assessing whether or not this is necessary to quickly locate a target. This reduced intricacy version of Seager's strategy also proves effective when comparing approaches. Similarly to Seager's, the simplified strategy also makes use of the target set, particularly that it can be divided into one set for each level in the tree, D_{j-1}, D_j, D_{j+1}. These sets are maintained differently and only D_{j-1} and D_{j+1} are of interest. As these are more generally tracked they will be referred to as U and L for the upper and lower level sets. The strategy operates in sequences of three probes. The root is the first probe of a sequence returning d_1, determining the level of t as level d_1 of the tree and defining a set D as level d_1. After the first probe, $U =$ *Parents*(D) and $L =$ *Children*(D), and after the second probe $U =$ *Parents*(U) and $L =$ *Children*(L). This creates a rough boundary which contains t but is far less accurate than in Seager's approach. The strategy is divided into four separate approaches. Assigning w and x to be the leftmost and rightmost nodes of U, and y and z to be the leftmost and rightmost nodes of L regardless of the nodes the sets contain, one of these four nodes will be probed throughout. For example, the *topLeft* version of the strategy will probe node w on the second and third probes of each sequence. The target will be better able to evade capture in this strategy due to the more relaxed approach of the probing player, as a result the approximate location of the target is reset with each initial probe of a sequence by probing the root. Section 5.3.2 discusses an MiniMax-based approach motivated by this strategy.

3.2 Target Strategies

In a theoretical setting it is assumed that a target with perfect knowledge plays optimally. While this is useful as it allows analysis of the worst-case capture time for probing strategies, the behaviour is not straightforward to implement. As such the target strategies outlined here are sub-optimal. The first simple target to be implemented is a completely random target that makes no judgements before moving to a particular node. As a result this strategy is considered a baseline for comparisons with the following two strategies. It seems logical to assume that a sensible target would move around the tree with some probability of visiting a certain node and therefore two probabilistic strategies have been implemented. The first probabilistic strategy considers the neighbourhood of the node currently occupied by the target and the degree of each node in the neighbourhood. Given a tree $T = (V, E)$ and node $v \in V$, the neighbourhood of v is the nodes contained in the set $N = \{v\} \cup \{u | (u, v) \in E\}$. In the case of a tree the neighbourhood of a node contains the node itself, the node's parent, and all of the node's children. The degree of v, $deg(v)$, is the number of edges v is incident to. The degree of all nodes in the neighbourhood is summed and each node $m \in N$ is assigned a probability equal to $\frac{deg(m)}{\sum_{n \in N} deg(n)}$. This then allows the target to move to some node based on its assigned probability. The second is a level-based probabilistic strategy which considers the fact that trees are the specific graph family being played on. In this strategy the target will move up or down the levels of the tree or remain at the current node with some probability. While these strategies seem sensible they are only informed by the structure of the tree rather than attempting to outplay the probing player. Section 3.3 discusses efforts to implement a target that is able to evade capture based on the probing player's moves and section 6 covers the limitations of the implemented strategies.

3.3 MiniMax

MiniMax, introduced in section 2.7, is a commonly used decision rule in game theory [39, 18, 13, 37, 30] where positive and negative players aim to maximise and minimise their scores respectively. A *search tree* is a rooted tree generated by the MiniMax algorithm where a node represents a move by one player, and all children of that node represent all possible moves that the other player may follow with. A winning move in the search tree is rewarded with $+\infty$ or $-\infty$ depending on the player, and a draw is represented as a reward of 0. As a two player game, it seems worth investigating the applicability of the decision rule to the localisation game. Developments were made throughout the project to implement a MiniMax model for the localisation game where both the target and probing player's moves were informed by the search tree. As there is no effective method of scoring a position in the localisation game, the Minimax algorithm implemented in this project generated entire search trees for small trees with ≤ 10 nodes in order to assess the suitability of the approach. Small trees were chosen as even for slightly larger trees generating an entire search tree is a very time consuming process, this is due to the large amount of moves available to the probing player in each round increasing the size of the tree a great deal. Once a search tree is generated each leaf node is analysed and the move correlating with the highest score for that player is chosen, this process is repeated for the parent level of these nodes resulting in a sequence of moves starting from the root that result in optimal play by both players.

4. IMPLEMENTATIONS

This section details the implementations of strategies outlined in section 3 with a brief discussion of time complexity. All code implementated throughout this project can be found at `https://github.com/Horse-Lips/Localisation_in_Trees`. Some time complexities given are intentionally brief due to the complex nature of the algorithms, for example in the outline of

the implementation of Seager's strategy. Section 4.1 covers more general functions used in all strategies and to handle the running of the game. Sections 4.2 and 4.3 outline the implementations of Seager's strategy and the simplified Seager strategy respectively, and section 4.4 details the implementations of the target strategies.

4.1 General Implementations

4.1.1 Tree Representation

Trees are represented as NetworkX [23] graph objects. As NetworkX does not provide a tree object directly, representations using Python [40] dictionaries are employed referred to as the *tDict* and *lDict* of the tree. A *NodeInfo* object is also maintained for each node in the tree, this structure maintains the level of the tree that the node is on, a pointer to the node's parent which is NULL for the root, and a list of pointers to the node's children. Each node is labeled from 0, the root of the tree, to $n-1$ for an n node tree.

> DEFINITION 4 (TDICT). *The tDict of a tree is a Python dictionary with a key for each node label, the value of this entry is the corresponding NodeInfo object for that node.*

> DEFINITION 5 (LDICT). *For a tree of height h, the lDict of the tree is a Python dictionary with a key for each level in the tree from* 0 *to* $h-1$. *The entry for each level is a list of pointers to nodes on that level of the tree.*

These representations allow the tree to be arranged as in the preprocessing step outlined by Seager as well as allowing algorithms to efficiently retrieve parents and children of nodes. The algorithm which creates these dictionaries is implemented recursively, given a tree it will create each dictionary from the root down and initialise a NodeInfo object for each node added to the *tDict*. This requires an iteration over each node of the tree resulting in an algorithm with linear time complexity in the size of the tree.

4.1.2 Probing and Target Sets

After determining some node v to probe, the function $probe(v)$ is the same for all strategies, returning the shortest path distance between v and t using NetworkX's shortest path length function. This makes use of Dijkstra's shortest path algorithm [17]. The target set is maintained as three lists of nodes, one for each level as in Seager's strategy. The sets are updated by an all pairs shortest path calculation using NetworkX's single source Dijkstra path length function with the probed node v as the source node. The function is shown in algorithm 1. The target set update function requires an iteration over each node of the tree, resulting in a time complexity of $O(n)$ for an n node tree.

4.1.3 Preprocessing and Hideout Detection

For the purposes of this project, all trees generated are hideout-free. This is implemented within the tree generation script according to lemma 1 of Seager's strategy [16], as well as their description of a hideout as in definition 3. The algorithm requires an iteration over each entry of the $tDict$, giving $O(n)$ for an n node tree. Trees are also reorganised to adhere to the arrangement described in section 3 with a node's leftmost child as its highest degree and its second-highest degree as its rightmost. This is performed during hideout detection and requires one iteration over the list of children of each node. This requires iteration over $n-1$ nodes throughout as the root of the tree will be ignored resulting in an $O(n^2)$ algorithm.

Data: A tree $T=(V,E)$, nodes $v, t \in V$, $d = probe(v)$, sets D_{j-1}, D_j, D_{j+1}, and a new level j.
Result: D_{j-1}, D_j, D_{j+1} updated according to value of d
begin
  $R \longleftarrow D_{j-1} \cup D_j \cup D_{j+1}$
  Assign D_{j-1}, D_j, D_{j+1} to be empty lists
  for $u \in V$ **do**
    if $u \in R \wedge dist(u,t) = d$ **then**
      if u *on level* $j-1$ **then**
        Add u to D_{j-1}
      else if u *on level* j **then**
        Add u to D_j
      else if u *on level* $j+1$ **then**
        Add u to D_{j+1}
    end
  end
end

Algorithm 1: Function for updating the target set.

Data: A tree $T=(V,E)$ represented as a $tDict$
Result: true if the tree contains a hideout, false otherwise
begin
  for *Each node entry* n *in the* $tDict$ **do**
    $C_n \longleftarrow tDict[n].children$
    $p_n \longleftarrow tDict[n].parent$
    if p_n *does not exist and* $|C_n| < 3$ **then**
      Skip node n
    else if p_n *exists and* $|C_n| < 2$ **then**
      Skip node n
    $C_{deg} \longleftarrow \{|tDict[c].children| + 1 \forall c \in C_n\}$
    $D_3 \longleftarrow |\{c \in C_{deg} | c >= 3\}|$
    if p_n *exists* **then**
      $p_{deg} \longleftarrow |tDict[p_n].children|$
      if p_n *has a parent* **then**
        $p_{deg} \longleftarrow p_{deg} + 1$
      end
      if $p_{deg} >= 3$ **then**
        $D_3 \longleftarrow D_3 + 1$
      end
    end
    if $D_3 >= 3$ **then**
      Return true
    end
    Set n's highest degree child as its leftmost
    Set n's second-highest degree child as its rightmost
    Return false
  end
end

Algorithm 2: Hideout detection algorithm.

4.2 Seager's Strategy

Mentioned previously was the fact that trees used for experimentation are generated hideout-free. As a result lemma 1 of Seager's strategy becomes redundant in the implementation here and has been omitted. Lemma 2 is implemented recursively, initially given w, z and recursively called with either $succ(w), z$ or $w, pred(z)$ as described in section 3.1.1. In the worst case this algorithm is $O(m)$ where $m = |\ Siblings(w,z)|$. As lemma 3 is a special case of lemma 2, the time complexity is the same in the worst case. It is difficult to analyse the worst case run time for lemma 4 and each of its cases due to the recursive nature of the implementation, there-

fore Seager's guaranteed *capture time* of $5 \cdot n$ [16] for an n node tree is provides an estimate for the run time of the entire strategy. As discussed in section 5 the run time of the algorithm is much lower in practice due to low capture time. The $tDict$ representation of trees allows the *Siblings*(w, z) function to run in constant time by retrieving the list of children of the parent of w from the dictionary. The *Children*(u, v) function is $O(m)$ where $m = |\{u, ..., v\}|$ as this requires iteration over the set $\{u, ..., v\}$ and retrieving each node's children from the $tDict$.

4.3 Simplified Seager

The simplified Seager approach operates by counting the current move number, expanding the target location sets when necessary, and then making the given move by retrieving the node to probe from the necessary list. These operations are possible through exploiting the earlier-created $lDict$. Upon the initial probe, the root of the tree, the set D_j can be obtained as level d of the tree where d is the distance returned by the probe. For the second probe, sets U and L can be obtained in $O(m)$ where $m = |D_j|$ by iterating over each node in D_j and adding its parents and children to U and L respectively. Finally for the third probe the sets U and L can be updated in $O(|U| + |L|)$. Each node in U is iterated over and its parents are added to the new set U, and then a similar process is carried out to update L by adding the children of each node to the set. This implementation of the game is separate from Seager's which requires precise sets of nodes to be probed in sequence. In this implementation, both the probe and target are defined as move generation functions passed to a class handling the game. This class is also given a NetworkX tree object and handles calling the necessary move generation functions. The class handles win detection which is determined when there is only a single node in the target and makes use of general functions such as $probe(v)$.

4.4 Target Strategies

Target movement functions are implemented in a similar manner to one another. Given the tree as a NetworkX object, a $tDict$, an $lDict$, and the node currently occupied by the target, the function will follow its defined set of rules to generate the next move for the target. These functions all operate in constant time as a uniform random number is generated and nodes can be retrieved by simply indexing the $tDict$ or $lDict$ and the list of nodes obtained from each. The target move generation function is given when initialising either the general localisation class or the Seager class. This move generation function is called each time a node is probed and there are no previously defined target moves for the player to make, as a list of predefined target moves can be given initially.

5. RESULTS

This section covers the details of the experiments carried out in this project. Section 5.1 explains the experimental setup and the process used to generate random hideout-free trees on which games are played. Section 5.2 shows the results of these experiments with an analysis of capture times and target win percentages in the case of the simplified approach. This project focuses on the notion of capture time in practice which differs from the more precise definition of theoretical capture time. In this setting the capture time is the mean number of rounds taken for a particular probing strategy to locate a particular target strategy as described in section 5.1. Section 5.2.3 discusses the run times of the algorithms used, and finally section 5.3 details the findings made when developing a MiniMax model for the localisation game. Experiments were carried out on a personal machine using a 12-core 4Ghz processor and 16GB RAM.

5.1 Tree Generation

A tree generation script was used to generate hideout-free trees for experimentation. Each tree was generated using NetworkX's random_tree function which generates a random Prüfer sequence [31] which is then converted into a tree. A Prüfer sequence is a labelling of an n node tree as a sequence of length $n-2$. For $1 \leq i \leq n-2$ the leaf with smallest label is removed and the label becomes the ith element of the sequence. An example is given in figure 2. The NetworkX algorithm selects a random tree uniformly from a set of generated n node Prüfer sequences. Lemma 1 of Seager's strategy [16] is used for linear time hideout detection within the tree generation script. 146 hideout-free trees were randomly generated ranging from 3 nodes to 150 nodes and Seager's tree[6] is used as the tree with 103 nodes. Section 6 discusses difficulties in generating larger hideout-free trees.

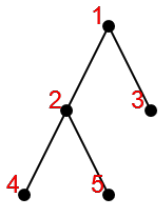


Figure 2: A small tree with Prüfer sequence $\{2, 2, 1\}$.

5.2 Seager and Simplified Seager

The simplified Seager strategy has four different variants described in section 3.1.2, probing either the leftmost or rightmost of either U or L. Each variant will be referred to as *tLeft*, *tRight*, *bLeft*, and *bRight* and Seager's strategy will be referred to as *Seager*. There are three different target strategies, a random target, a target that moves to a node with probability based on its degree, and a target that moves up or down the levels of a tree with some probability. These will be referred to as *Rt*, *Pt*, and *LBt* respectively. In the following experiments LBt has a 35% chance to move to the node's parent, a 50% chance to move to a random child of the node, or a 15% chance to remain at its current node. Each combination of probe and target strategy are used on each tree, with each node used as a target starting position 25 times. For each tree, each node's mean capture time and run time are recorded over these 25 games and the tree's mean capture and run time is calculated as the mean of node mean. In a theoretical setting a target's starting position would be optimal so as to allow the target to evade capture for as long as possible, these varied target starting positions aim to remedy sub-optimal target movement across games. An upper bound on capture time can also be guaranteed in a theoretical setting and Seager gives a guaranteed capture time for their strategy as $5 \cdot n$ for an n node tree. As target play is not optimal and neither are the simplified Seager probing strategies there are some situations where the target may win even with the absence of hideouts. Seager notes that $5 \cdot n$ seems an unreasonably high upper limit and Behague et al. showed that trees are well-localisable[7] [10], therefore for the purposes of these experiments it has been reduced. If the target is able to evade capture for n rounds then a target win is reported.

5.2.1 Simplified Seager

[6] See figure 4 [35].

[7] Recall a well-localisable graph has capture time linear in the size of the graph.

Figure 3 shows the results obtained using the tLeft version of the simplified Seager strategy. Figures 5, 6, 7 can be found in the appendix and show the results for tRight, bLeft, and bRight respectively. These have been removed from this section as all results given by simplified Seager approaches are incredibly similar to one another. This is most likely caused by the simplicity and similarity of the approaches. Each version of the strategy features a very relaxed approach to probing which is only altered depending on the position of the target or version chosen. The figure shows a steady increase in capture time as tree size increases, eventually showing very large capture times for larger trees. This is the expected behaviour of all strategies as probe accuracy is expected to decrease with increased tree size. Accuracy in this sense refers to the number of possible target locations after a probe, where higher accuracy implies there are fewer possible locations. This coupled with the relaxed nature of the strategy leads to large capture times even for smaller trees. This contrasts with Seager's strategy where it is the simplicity of target movement strategies instead that leads to the results shown. There are also some large increases and decreases in capture time in the trees with between 100 and 110, and between 120 and 130 nodes, where it is believed that this is caused by the underlying structure of the trees[8].

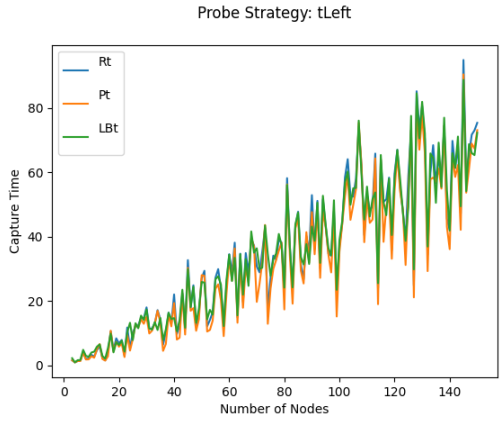


Figure 3: Mean capture times with increasing tree size using the tLeft strategy.

As a more approximate and relaxed probing strategy, even a simple target is expected to win some amount of games. A second experiment was conducted where each probing strategy played against each target. Games were played on the randomly generated trees using each node as a target starting position once and the percentage of target wins was recorded for each pair of strategies. Table 1 shows that the target wins between 20 and 30 percent of games with some variation caused by the target's random or probabilistic nature. This seems very reasonable considering the approximate way in which the probing player keeps track of the target. Interestingly, LBt has the highest win percentage against all probing strategies. While this is the only strategy that makes use of the structure of the tree itself, this only shows that the target performs well against a more simplified probing strategy. In fact, figure 4 shows that against a more precise strategy that LBt appears to have a lower mean capture time overall.

5.2.2 *Seager's Strategy*

Figure 4 shows the results of Seager's strategy against each type of target. Compared with the simplified Seager approach, this strategy shows a sharper increase in capture time for trees under 60 nodes followed by a slower increase for larger trees. The increase in capture time becoming slower as trees become larger is unexpected as the accuracy of probes is thought to decrease. Capture times are also incredibly low even for the largest trees tested. There are two reasons for this behaviour, mainly the simplicity of target movement strategies, described in section 3.2. Seager's strategy maintains very precise sets in order to track an approximate target location throughout the game, allowing target capture incredibly quickly. The simple nature of employed target strategies contributes to this, as suboptimal target movement could lead to immediate capture for many of the components of the strategy. For example in the context of lemma 2 of Seager's strategy, discussed in section 3.1.1, a target moving anywhere but a sibling or child of the sibling set being examined will be captured immediately. An alternative factor that may contribute to this behaviour is the underlying structure of the trees. Similarly to the simplified Seager approach, this underlying tree structure may contribute to capture time spikes seen in trees with roughly 45 and 100 nodes. The structure of trees is further discussed in section 6. Seager's approach is viable theoretically as it is able to locate an optimal target within a finite number of rounds. As a result the probe was able to capture the target in all games played. Seager's strategy gives a guaranteed capture time of $5 \cdot n$ for an n node tree [16], and these experiments lowered this limit using a maximum value of n. The strategy greatly exceeded even this low capture time limit, which further justifies Seager's claim that $5 \cdot n$ is much larger than necessary. As discussed earlier, this is largely due to the simplicity of the target strategies employed, but shows that suboptimal target movement is unsuitable against more precise probing strategies.

	Rt	**Pt**	**LBt**
tLeft	27.72%	24.73%	30.25%
tRight	26.95%	22.68%	29.61%
bLeft	24.17%	19.80%	25.67%
bRight	25.50%	21.05%	26.90%

Table 1: Target win percentages for each simplified probing strategy.

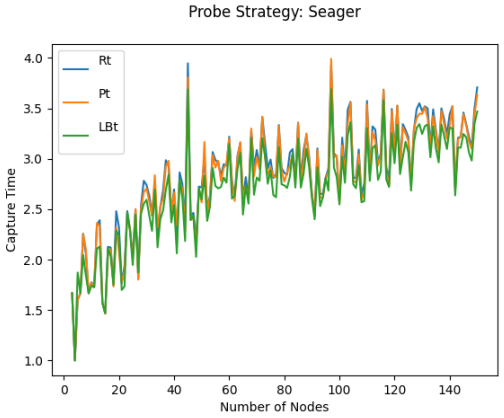


Figure 4: Mean capture times with increasing tree size using Seager's strategy.

5.2.3 *Run Time Analysis*

Table 2 shows the run time of each probing strategy algorithm when playing against each type of target. Mean run times are recorded as the overall mean of each node's mean run time. Recall that a node's mean run time is calculated as the mean run time for a given

[8]Recall that only one tree of each size is used.

	Rt			Pt			LBt		
	Min	Mean	Max	Min	Mean	Max	Min	Mean	Max
Seager	5μs	0.4ms	11ms	19μs	0.4ms	15ms	4μs	0.5ms	12ms
tLeft	20μs	6ms	29ms	2μs	6ms	28ms	3μs	6ms	27ms
tRight	3μs	6ms	33ms	4μs	5ms	27ms	19μs	5ms	26ms
bLeft	4μs	6ms	33ms	19μs	5ms	28ms	20μs	5ms	26ms
bRight	3μs	6ms	34ms	3μs	5ms	26ms	19μs	5ms	26ms

Table 2: Mean run times for probe strategies in milliseconds.

strategy pair playing 25 games using that node as a target starting position. All algorithms perform very efficiently with the largest mean run time recorded by the simplified Seager approach at 6 milliseconds. Seager's strategy appears to outperform all strategies against every type of target with the lowest of all run times, which are also incredibly low. These lower run times are directly correlated with the low capture time of the strategy. As discussed earlier, the basic nature of target movement strategies leads to an incredibly quick capture time for such a precise algorithm, which in turn leads to low run times due to the fact that not many components of the strategy are utilised. Aside from this the run time is still considered very efficient considering the intricacy of the strategy. Maximum run times follow a similar trend where each of the simplified Seager approaches perform very similarly to one another, and with Seager's strategy outperforming each by a large margin. Minimum run times however appear to show that occasionally simplified strategies perform similarly to Seager's. This is likely caused by the target being located almost immediately in smaller trees, however when dealing with such small values there could also be inconsistencies in the precision of times.

5.3 MiniMax

This section discusses findings made during the investigation of a MiniMax model for the localisation game. The algorithm was tested with very small trees generating an entire search tree, however the approach quickly became problematic. Where target movement is concerned the approach appears to generate optimal moves using the search tree. With this model the target is able to analyse moves that will lead to an eventual loss and make the best judgement based on its current position. Not only does this allow the target to play optimally against any strategy, but allows the target to select an optimal starting position enabling it to evade capture for as long as possible. Problems begin to occur when informing the probing player using the search tree. Due to the way that the search tree is constructed, each player receives a score based on its moves until a leaf node is reached that either indicates a win for one of the players or a draw. As the target makes the first move, each target starting position will be the root of each search tree generated. The probing player makes the next move, constructing level 1 of the search tree. As scores are calculated for the probing player's moves, the player is informed of its optimal move given the target's initial move. This allows the probing player to win on its first move by probing the first node encountered that enables capture of the target in a single probe. This is always possible by probing the node that the target occupies directly, but can also be achieved by probing the first node that results in a target set of size 1. There are three possible solutions that address this problematic behaviour, however each comes with its own issues.

5.3.1 Allowing External Probing Strategies

The first solution is to adapt the MiniMax algorithm to function with external strategies, such as Seager's strategy. In this setting only the target would have its moves informed by the search tree enabling existing strategies to be tested against an optimally playing target. This approach has its own problems to consider, namely that the target's moves are informed by a search tree that assumes optimal play by the probing player. This assumption would lead to a sub-optimal target that would assume it would lose after the probe plays its next move, however alterations to search tree construction could remedy this issue.

5.3.2 Reduced Probe Moveset

Another possible solution that was alluded to in section 3.1.2 is informing a reduced moveset probing player with a MiniMax search tree. For example the probing player would have its moves informed by the search tree, however the moves available to the player would be reduced to some subset of nodes in each round, for example the moveset available to the simplified version of Seager's strategy. This would enable the probing player to pick the best of its available moves in each round without winning immediately, however this solution is also very problematic. If the moveset were to be reduced to that of the simplified Seager approach, there is likely a subset of nodes of a large enough tree that enables the target to evade capture indefinitely. If the moveset is too large then a similar problem to the basic MiniMax approach occurs where the probing player is too powerful. This leads to a new problem of balancing the size and nodes included in the probing player's moveset in each round.

5.3.3 Reduced Probe Information

Alternatively reducing the information available to the probing player is another possible solution. Two versions of this method were attempted during this project and appear to remedy the problem of immediate target capture. Both solutions are reminiscent of the centroidal localisation game [9] in which the probing player may only compare distances between successive probes or is informed of the target being probed directly. The first approach allows the probing player to compare the distance returned by the current probe with the distance of the previous probe. The player receives a score which increases as the distances become smaller with a score of 0 if the probes return the same distance and an increasingly negative score if the distance becomes larger. The second approach is similar to the first, however the player is allowed to compare the distance returned by the current probe to all previous probes. A score is calculated which increases with the number of prior probes which returned a greater distance than the current probe. In both cases the results were almost identical in that the probing player placed probes which gradually located the target by eventually probing it directly. This is due to the fact that the player can only assess whether or not probes are becoming closer to the target's location, and in many cases probing nodes in the neighbourhood of the target would not locate it. In both cases it seems that the probe is still too powerful with a MiniMax-based approach.

6. CONCLUSIONS

The results shown in section 5 show that Seager's strategy [16] out-

performs the simplified probing strategy against all types of target considered while greatly exceeding the theoretical guaranteed capture time. The results of both probing strategies also support the findings of Behague et al. that trees in general are well-localisable[10]. While run time is directly correlated with capture time, it is also shown that the run times for Seager's strategy are low. It is unclear how Seager's strategy will perform in trees larger than 150 nodes as this was the largest tree size studied in this project. This can be seen as a limitation of the project as with larger trees the capture time could gradually increase further. Future work could attempt to generate larger hideout-free trees in order to analyse this behaviour, however this is problematic due to the exclusion of hideouts limiting the branching factor of larger trees. Another limitation of this project is the lack of analysis of the underlying structure of trees experimented on. It was noted that sudden dramatic changes in capture time graphs could be the result of the underlying tree's structure. Future work could investigate the structure of trees on which the game is played in order to assess whether or not certain strategies perform better under certain conditions. As for target strategies, those outlined in section 5 are very simple with no knowledge of the game or probing player's moves. While these targets prove useful in comparing strategies to one another, a better target with game knowledge could provide more informative results in terms of worst-case capture time of probing strategies. With regard to this, the work conducted towards developing a MiniMax approach shows promise. It seems that modification of the MiniMax algorithm could allow use with external probing strategies and target play could become closer to optimal, allowing an assessment of worst-case scenarios for probing strategies. An issue that may arise with such an approach, however, is the time-consuming nature of creating full search trees for larger graphs. After the target's initial move as the tree's root, each level representing the probing player's moves adds n children to each leaf in the search tree for an n node tree. Each possible target move adds $deg(t)$ nodes as children to each of these leaves and so on. Given a binary tree with n nodes and branching factor of 2 there are n possible probe moves and 4 possible target moves, t, t's parent, and t's 2 children, in each round. As trees are well-localisable this requires n rounds, and therefore generating a search tree with $(4 \cdot n)^n$ nodes. Lastly, there are several solutions to the immediate target capture problem in a MiniMax setting. Attempted in this project was a reduced information probing player which showed promising results, although the player still seems far too powerful. It still remains to find a good way of reducing the available information to the probing player in order for a fair game to take place. A solution that was not attempted was a reduced moveset probing player. There are several apparent problems with generating a moveset of adequate size and each of these has been discussed in section 5.3. Different sizes and structures of trees makes it seem unlikely that reduced movesets can be constructed by hand, as these properties will dictate a fair moveset in each round. It is not clear if there is a way of algorithmically determining movesets, however they could possibly be generated using subsets of the metric dimension of trees. Investigation into determining fair movesets for the probing player could lead to a promising MiniMax model for the localisation game.

Acknowledgments. Many thanks to Dr. Jessica Enright for their continued support and guidance while supervising this project.

7. REFERENCES

[1] N. C. Behague, A. Bonato, M. A. Huggan, T. G. Marbach, and B. Pittman. The localization capture time of a graph. *Theoretical Computer Science*, 911:80–91, 2022.

[2] E. A. Bender and S. G. Williamson. *Lists, decisions and graphs*. S. Gill Williamson, 2010.

[3] J. Bensmail, D. Mazauric, F. Mc Inerney, N. Nisse, and S. Pérennes. Sequential metric dimension. *Algorithmica*, 82(10):2867–2901, 2020.

[4] A. Bonato, M. A. Huggan, and T. Marbach. The localization number and metric dimension of graphs of diameter 2, 2020.

[5] A. Bonato, M. A. Huggan, and T. G. Marbach. The localization number of designs. *Journal of Combinatorial Designs*, 29(3):175–192, 2021.

[6] A. Bonato, M. A. Huggan, and T. G. Marbach. Progress on the localization number of a graph. *arXiv preprint arXiv:2103.10587*, 2021.

[7] A. Bonato and W. B. Kinnersley. Bounds on the localization number. *Journal of Graph Theory*, 94(4):579–596, 2020.

[8] A. Bonato, T. G. Marbach, M. Molnar, and J. Nir. The one-visibility localization game. *arXiv preprint arXiv:2301.03534*, 2023.

[9] B. Bosek, P. Gordinowicz, J. Grytczuk, N. Nisse, J. Sokół, and M. Śleszyńska-Nowak. Centroidal localization game. *arXiv preprint arXiv:1711.08836*, 2017.

[10] B. Bosek, P. Gordinowicz, J. Grytczuk, N. Nisse, J. Sokół, and M. Śleszyńska Nowak. Localization game on geometric and planar graphs. *Discrete Applied Mathematics*, 251:30–39, 2018.

[11] J. Boshoff and A. Roux. The localization game on cartesian products. *Discrete Applied Mathematics*, 305:247–259, 2021.

[12] A. Brandt, J. Diemunsch, C. Erbes, J. LeGrand, and C. Moffatt. A robber locating strategy for trees. *Discrete Applied Mathematics*, 232:99–106, 2017.

[13] F. Brandt, M. Brill, and W. Suksompong. An ordinal minimax theorem. *Games and Economic Behavior*, 95:107–112, 2016.

[14] J. Cáceres, C. Hernando, M. Mora, I. M. Pelayo, M. L. Puertas, C. Seara, and D. R. Wood. On the metric dimension of cartesian products of graphs. *SIAM journal on discrete mathematics*, 21(2):423–441, 2007.

[15] J. Cáceres, C. Hernando, M. Mora, I. M. Pelayo, M. L. Puertas, C. Seara, and D. R. Wood. On the metric dimension of cartesian products of graphs. *SIAM journal on discrete mathematics*, 21(2):423–441, 2007.

[16] J. Carraher, I. Choi, M. Delcourt, L. H. Erickson, and D. B. West. Locating a robber on a graph via distance queries. *Theoretical Computer Science*, 463:54–61, 2012. Special Issue on Theory and Applications of Graph Searching Problems.

[17] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959.

[18] D.-Z. Du and P. M. Pardalos. *Minimax and applications*, volume 4. Springer Science & Business Media, 1995.

[19] A. Dudek, S. English, A. Frieze, C. MacRury, and P. Prałat. Localization game for random graphs. *Discrete Applied Mathematics*, 309:202–214, 2022.

[20] A. Dudek, A. Frieze, and W. Pegden. A note on the localization number of random graphs: Diameter two case. *Discrete Applied Mathematics*, 254:107–112, 2019.

[21] F. Foucaud, R. Klasing, and P. J. Slater. Centroidal bases in graphs. *Networks*, 64(2):96–108, 2014.

[22] M. R. Gary and D. S. Johnson. Computers and intractability: A guide to the theory of np-completeness, 1979.

[23] A. Hagberg, P. Swart, and D. S Chult. Exploring network structure, dynamics, and function using networkx. Technical report, Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2008.

[24] F. Harary and R. Melter. On the metric dimension of a graph. *Combinatoria, vol. 2*, 1976.

[25] J. Haslegrave, R. A. Johnson, and S. Koch. Subdivisions in the robber locating game. *Discrete Mathematics*, 339(11):2804–2811, 2016.

[26] J. Haslegrave, R. A. Johnson, and S. Koch. Locating a robber with multiple probes. *Discrete Mathematics*, 341(1):184–193, 2018.

[27] L. Lichev, D. Mitsche, and P. Pralat. Localization game for random geometric graphs, 2021.

[28] L. LOVÁSZ and D. MICHAEL. Plummer: Matching theory. *Annals of Discrete Mathematics*, 29, 1986.

[29] T. Parsons. Pursuit-evasion in a graph. *Theory and Applications of Graphs*, pages 426–441, 1976.

[30] T. Parthasarathy. On games over the unit square. *SIAM Journal on Applied Mathematics*, 19(2):473–476, 1970.

[31] H. Prüfer. Neuer beweis eines satzes über permutationen. *Arch. Math. Phys*, 27(1918):742–744, 1918.

[32] N. Robertson and P. Seymour. Graph minors xxiii. nash-williams' immersion conjecture. *Journal of Combinatorial Theory, Series B*, 100(2):181–205, 2010.

[33] G. Sabidussi. 10.1007/bf01162967. *Math. Z*, 72:446–457, 1960.

[34] S. Seager. Locating a robber on a graph. *Discrete Mathematics*, 312(22):3265–3269, 2012.

[35] S. Seager. Locating a backtracking robber on a tree. *Theoretical Computer Science*, 539:28–37, 2014.

[36] S. M. Seager. A sequential locating game on graphs. *Ars Comb.*, 110:45–54, 2013.

[37] M. Sion. On general minimax theorems. *Pacific Journal of mathematics*, 8(1):171–176, 1958.

[38] P. J. Slater. Leaves of trees. *Congr. Numer*, 14(549-559):37, 1975.

[39] J. v. Neumann. Zur theorie der gesellschaftsspiele. *Mathematische annalen*, 100(1):295–320, 1928.

[40] G. vanRossum. Python reference manual. *Department of Computer Science [CS]*, (R 9525), 1995.

8. APPENDIX

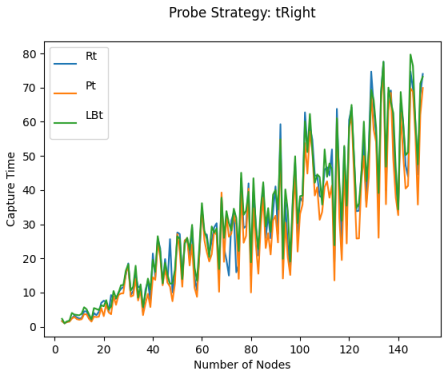


Figure 5: Mean capture times with increasing tree size using the tRight strategy.

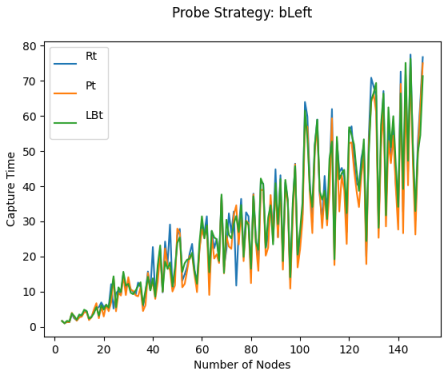


Figure 6: Mean capture times with increasing tree size using the bLeft strategy.

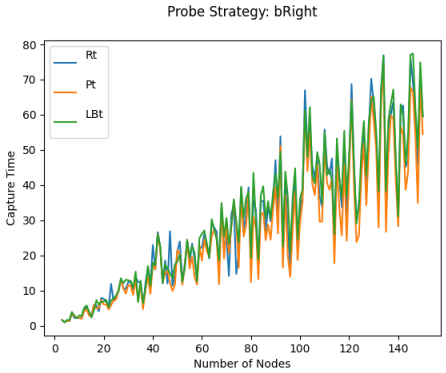


Figure 7: Mean capture times with increasing tree size using the bRight strategy.