Dissertation zur Erlangung des Doktorgrades der Technischen Fakultät der Albert-Ludwigs-Universität Freiburg im Breisgau

Collaborative Graph Exploration

Institut für Informatik
Technische Fakultät
Albert-Ludwigs-Universität Freiburg

Christian Ortolf

April 2016

Collaborative Graph Exploration

Christian Ortolf

Dissertation zur Erlangung des Doktorgrades der Technischen Fakultät der Albert-Ludwigs-Universität Freiburg im Breisgau

Dekan: Prof. Dr. Georg Lausen

Erstgutachter: Prof. Dr. Christian Schindelhauer (Universität Freiburg)
Zweitgutachter: Prof. Dr. Fabian Kuhn (Universität Freiburg)

Datum der Disputation: 18. Juli 2016

Zusammenfassung

In der vorliegenden Arbeit wird die Exploration von einer Landschaft mit Hilfe von Teams aus Robotern diskutiert. Die Landschaft wird dabei von einem Graphen modelliert, an dessen Kanten die Roboter in einem rundenbasierten Modell entlanglaufen, mit dem Ziel alle Knoten des Graphen zu besuchen. Ein Algorithmus wählt dazu für jeden Roboter in jeder Runde eine anliegende Kante aus, an der dieser entlanglaufen kann, um zu einem neuen Knoten zu gelangen.

Die besondere Schwierigkeit, im Gegensatz zum Problem des Handlungsreisenden mit mehreren Handlungsreisenden (mTSP), ist dabei, dass das betrachtete Team weiter eingeschränkt wird.

In unserem ersten Szenario (Exploration) wird dem Team der Überblick über den Graphen verwehrt, so dass der Algorithmus alle Entscheidungen auf dem bereits besuchten Teil des Graphen gründen muss. Wir zeigen für dieses Szenario neue untere Schranken auf Gittergraphen mit rechteckigen Hindernissen. Zudem erweitern wir die unteren Schranken für diese Gittergraphen und auch auf Bäumen damit sie auch gegen randomisierte Algorithmen bestehen. Des Weiteren zeigen wir effiziente Algorithmen für die Erkundung der Gittergraphen als auch für Bäume. Um die Untersuchung mit empirischen Ergebnissen zu ergänzen, zeigen wir noch Experimente auf sogenannten Kammbäumen, eine für viele Algorithmen schwierige Art von Graph.

Als weiteres Explorationsszenario behandeln wir Roboter, die zwar eine Karte besitzen, jedoch nicht miteinander kommunizieren können (unaware cleaning). Dies schließt auch die Wahrnehmung anderer Roboter aus. Dabei ist es nicht nur interessant alle Knoten des Graphen einmal abzulaufen, sondern auch diese immer wieder zu besuchen. Dazu müssen Roboter sich so bewegen, dass ihre Bewegung kompatibel mit der der anderen Roboter ist. Gemeinsam soll so ein effizientes Durchsuchen oder Reinigen einer durch einen Graphen modellierten Landschaft ermöglicht werden. Wir zeigen für dieses Szenario einen Algorithmus der asymptotisch optimal Gittergraphen (ohne Hindernisse) reinigen kann. Zudem zeigen wir auch wie ein solches Reinigungsteam eine komplexe Landschaft, die durch einen beliebigen Graphen modelliert wird, angehen kann. Am Ende evaluieren wir noch empirisch unsere Algorithmen auf dem Gittergraphen um eine Intuition zu verschaffen für die Effizienz und Probleme dieser Algorithmen.

Abstract

This thesis discusses the exploration of a landscape by a team of robots. The landscape is for that purpose modeled by a graph, along whose edges robots may travel in a round-based model with the goal to visit all vertices of the graph. For this purpose an algorithm chooses for each robot in each round an incident edge to traverse to reach a new vertex with that robot.

The special challenge, in contrast to the traveling salesman problem with multiple salesman (mTSP), is, that the considered team is being further restricted.

In our first scenario (exploration) the team is forbidden to see an overview of the graph, such that the algorithm must base all its decisions on the already visited part of the graph. We show for this scenario new lower bounds on grid graphs with rectangular obstacles. In addition we extends the lower bounds for these grid graphs and for trees to hold up against randomized algorithms. Furthermore we show efficient algorithms for the exploration of grid graphs as well as for trees. To complete our analysis we present experiments on so called comb trees, a class of graph that forces several algorithms into their worst case behavior.

As a second exploration scenario we examine robots which indeed have a map of the graph, but may not communicate with each other (unaware cleaning). This also excludes the perception of other robots. For this scenario it is not only interesting to visit all nodes of the graph once, but to indefinitely repeat visiting them. To achieve this robots have to move in a way, such that their movement is compatible with that of other robots. Collectively this should enable them to efficiently search or clean a graph modeling a landscape. We prove for this scenario an algorithm cleaning grid graphs (without obstacles) in asymptotic optimal time. Additionally we show, how such a cleaning team can tackle a general graph modeling a complex landscape. Finally we evaluate empirically our algorithms on the grid graph to provide an intuition for the efficiency and problems of these cleaning algorithms.

Acknowledgments

Foremost I would like to thank my advisor Christian Schindelhauer for the opportunity to work at his chair. Hundreds of hours of conversation and discussion with him have left an imprint on me that can hardly be described with a few words. With no topic ever being of limits and the time he sacrificed for discussions, this created a remarkable work climate enabling me to thrive.

Many thanks go to past and present colleagues Faisal Aslam, Amir Bannoura, Thomas Janson, Sven Köhler, Norbert Küchlin, Kerstin Pfeiffer, Arne Vater and Johannes Wendeberg for many open-minded discussions about each others research, many shared coffee and lunch breaks, given help when ever needed and generally being part of the positive work climate.

Last but not least goes a thanks to Sarah Heitzler, Robert Jakob, Thomas Janson, Matthias Keil and Julia Kolter who proofread this thesis.

Contents

1 Introduction

When thinking of the word exploration, pictures form in one's mind of Christopher Columbus sailing over the ocean towards India just to discover a continent that was unknown to the people in Europe. A picture of the expedition of a young Alexander von Humboldt forms navigating his boats upstream on the Orinoco River, discovering new animal species on the way.

Exploration as an objective of research may appear at first glance to be outdated.

With satellites orbiting and watching the earth, exploration seems to be a task of the past. The days of ships sailing over an ocean to discover new continents are over. Satellites and airplanes produce maps of the earth easily accessible to anyone with a smartphone. On the other hand researchers are still venturing to remote places looking for yet undiscovered flora and fauna. But even mapping data of an urban area may simply not be precise enough for every task required, e.g. steering self-driving cars or robotic lawnmowers. Maps might be inaccessible to a robot as internet connectivity could be considered not permissible under a strict security model or unavailable for a private indoor environment such as a factory floor or a chamber cleaned by a robot vacuum cleaner. In addition the environment could often be changing, because new furniture is added to a room or a child's toys are left lying around obstructing a formerly accessible path of a cleaning robot. The environment could also be severely changed, for example a team of robots searching for a missing person after an earthquake, indoor, outdoor or underground, will be forced to work without a reliable map, as an earthquake can change any environment in a hardly foreseeable way.

Last but not least, the exploration of space has just begun and while writing this the space probe New Horizons has just sent back high resolution images from the dwarf-planet Pluto. The mars rover Curiosity is still exploring the surface of Mars which is not as well mapped out as the planet we are living on.

All these endeavors to further knowledge of the environment can be categorized as forms of exploration. Many of the named examples could also be categorized as research, but not all. Nobody would call a vacuum cleaner's quest for covering a room thoroughly research, but to a cleaner, that is oblivious to yesterday's cleaning of the same room, every run is a new exploration. If we look up Webster's simple explanation of the word exploration we get "the act or an instance of searching through or into" [exp16]. This is not to far of from the source of the word research which Webster provides as from middle french "recerche - to go about seeking" [res16].

The similarity between the two words is mind-boggling, especially if we start to interpret environment to be searched in a wider sense.

For example the world wide web forms a graph containing text or knowledge that can be explored by a software agent: Such a web crawler has to explore the web-graph to find all available data and make it available via an index for end users. If a human does the same, following links and interpreting the found text, it may as well be called research as it is an exploration.

It does not end with exploring data. Even computation itself can be viewed from the perspective of an explorer. The computation tree defined by a non-deterministic Turing machine (NTM) and its input-band can be considered an explorable environment. An agent similar to a deterministic Turing machine, but with additional capability to revert to the former state (Reversible TM [Ben73]), can simulate the NTM by exploring the whole computation tree of the NTM. However, in order to do this, to find an accepting state, the whole computation tree has to be explored in the worst case.

In the web as well as in the real world, teamwork or parallelism becomes a necessity for large environments or problem sizes. While a single robot may be able to explore and clean a house, it would be near impossible or at least uneconomic to build a single robot capable of cleaning a whole city on its own or building a computer capable of crawling the whole world wide web by itself. In a similar way our agent exploring the computation tree would require time exponential in the depth of the tree, while an exponentially large set of agents could explore the whole tree in polynomial time.

Coordinating the team is necessary to not explore the same part of the web-graph multiple times or distribute the cleaning robots evenly over an area to avoid any unwanted duplication of effort arising from cleaning an area multiple times or walking to an area with too many robots.

Having a map of the cleaning or mowing area or knowing how many many web pages are to be expected on every website, would allow us to distribute the work evenly. At the beginning the team could divide the work load fairly. Given just a map and an enumerated team no communication is necessary, because every team member could separate the work identically and then assign itself the workload according to its own number. Without a map, communication can be used to distribute the work, call for more help and repeatedly redistribute the team-members if the last distribution leaves a lot to be desired.

But for very large teams the communication to coordinate robots becomes in itself a hard task. In extreme environments, such as tunnels, robots may be unable to communicate, distances in cities may be too large to communicate point-to-point with antennas or a present communication infrastructure and protocols may be unsuitable for a large crowd of robots. Also communication could become unaffordably expensive compared to the rest of the robot, e.g. if robots are small and cheap or in an environment that needs very sophisticated communication equipment to enable communication at all, for instance robots operating underwater or a robot digging through the ground.

Another motivation to consider working without communication is, that the communication between robots can pose a security threat. In so called patrolling scenarios robots cover an area similar to cleaners, but are looking for intruders that are more intelligent than dust bunnies. In such a scenario, radio waves used for communication might allow an intelligent intruder to spot and surveil patrolling robots and therefore could ease intrusion into an area. Moreover, not just communication can be a problem, if patrolling robots know about each other, capturing one robot and reading out its storage could provide all information needed to evade the patrolling team. Even if robots simply move in some choreographed way, without communicating after the start of the patrolling, an attacker that only spots a few robots may be able to guess where other robots are likely to appear. Using unaware robots for patrolling would have the advantage that they are randomized, so no regular patterns can be exploited by an intruder. In addition, the lack of communication or need for visibility between each other could be a good model for robots, that try to be invisible to an intruder by being camouflaged and keeping radio silence. The absence of even the knowledge on how many other robots are in the field, makes such a robot near worthless for an intruder capable of capturing a robot for intelligence purposes.

Our Focus

In this thesis, our focus lies in team work for exploration. Our central question is how to efficiently use a large number of robots to parallelize the exploration of an area.

In Chapter 2 we look at the classical exploration without a map: The exploration for the purpose of creating a map. Robots will communicate somehow, but our main concern is not to handle the communication, but the question of how to distribute the robots in a way that guarantees a good runtime for the exploration. The goal is to answer the questions: How much does it help an exploration team, if we use more robots? Or what is the difference if you give a team twice as much time or twice as many team members? We answer these questions by showing the efficiency of such a team without a map, compared to a team of the same size that uses map of the area for making better choices in distributing its robots.

In Chapter 3 we look into organizing the exploring team with a map with as little communication as possible. We ask the question if we can efficiently organize an exploration of a known area in the sense of cleaning, mowing, searching or patrolling without making use of any communication at all. Neither direct communication, nor any form of indirect communication such as enumerating or getting a count of other team members or observing the movement patterns of other team members, will be allowed. Here we ask again the question of how much additional time is required, if the owner deploys a large team of unaware robots. We answer this by comparing the team's efficiency to that of a communicating team and we evaluate the performance for differing team and graph sizes.

If we expand our view, we can include a lot of types of work done by teams, but we also need to distinguish what should not be viewed as an exploration. We see exploration as work where the movement dominates any time-cost. For example, it is common for current vacuum cleaners to move at about the same speed while vacuuming as when not vacuuming. If the team has work to do in an area that dominates the cost for the movement to such an extent that any movement can be amortized with the cost for the work, it is for our purpose no exploration. We expect exploration to only consist of movement, therefore if work is needed in an area, it must be small enough to be amortizable by the movement, so we can ignore it.

No method of fast or instant travel is allowed or possible. For the cleaning robot this could mean to take a taxi to his next place of cleaning, if the speed of the taxi compared to cleaning is sufficiently fast, it can be considered instantaneous. Smartly redistributing robots in reaction to newly found territory, but also the changing cost for distributing robots to locations explored further away from each other is a central aspect of collaborative exploration. If the cost associated with instant travel is cheap enough to make it viable for any step, there is no longer a point in modeling the problem on a landscape. We could then distribute robots to workloads and redistribute robots as needed, ignorant of different costs between different balancing strategies. For example a web-crawler can jump to any web page of a static website just by requesting the URL it is interested in from the server. It doesn't have to go there by consecutively requesting linked documents that will finally lead it to the document of interest.

Modeling the Real World

To study these challenges of robot teams and evade the need for a real-world testbed with thousands of robots, a mathematical model is used. We will formally introduce this model later in Section 1.1. We represent robots as entities moving along the edges of a graph in a round-based model.

Hence, this allows us a different point of view for the quality of such algorithms than would otherwise be affordable. Asymptotic run times are not visible in testbeds with dozens or even less robots. Even the largest simulations we are capable of executing, with thousands of robots and millions of nodes, fail to demonstrate a difference between polylogarithmic and small polynomial run times.

Why model exploration with graphs? The discretization of real terrain into a graph depends on the way a robot perceives the environment. In case of a web-crawler the environment is already discrete and representation as a graph comes naturally.

For the real world, this discretization depends on the perception capabilities of the robot. If a satellite picture from a landscape is given, shape analysis can be used to extract a graph from the pixels or contours in a bitmap, a complex topic of its own (for a survey we refer to [Lon98]).

For navigation, the positioning and environment information has to be extracted from sensor data of a robot and has somehow to be translated into a format in which algorithms can work efficiently. This mobile robot localization problem is a key problem for robotics and will not be discussed in this thesis. We do assume that any robot is capable of perfectly locating itself in any environment.

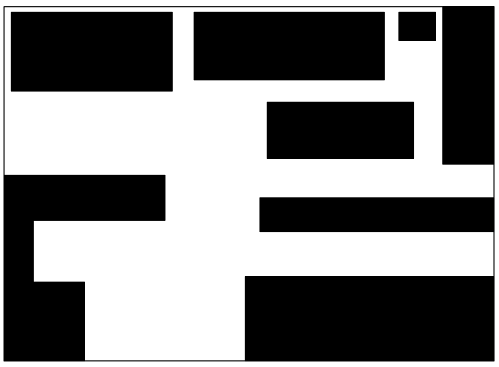
Since movement happens for non-flying robots in a two dimensional environment, it is natural to use a two dimensional abstraction of the environment. A natural description for such an environment is to utilize geometrical forms, i.e. an Euclidean plane in combination with simple polygons to approximate obstacles, for instance in [DKP91, AKS99]. Sight can be used to explore such a plane with respect to the obstacles. But arguing proofs with geometry instead of graphs is complicated. For implementation in a simulation we face a similar problem. A discrete model kept as graph is simpler to compute on, than a complex geometrical environment.

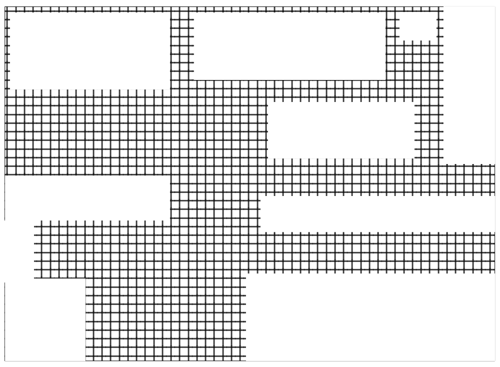
One way to translate a polygon landscape to a graph is to triangulate it. Every triangle can then be interpreted as a vertex of a corresponding graph that is to be explored, with edges connecting the nodes that represent triangles adjacent to each other. A robot on a node representing a triangle can see then the whole area covered by the triangle if its sight is unlimited. This is somewhat similar to the solution to the art gallery problem [Chv75], where a polygon-shape must be be covered with star-shaped polygons. Using coloring to optimize the number of nodes in the graph is a possibility to gain a constant factor, but this omits the problem of which vertices, representing star-shapes, should be connected.

Polygons are a good discretization for robots capable of sight and transforming these into graphs may preserve a geometrically formulated problem. For example, seeing everything of a landscape as a problem is preserved, but gradually perceiving more of a landscape while walking around a corner is not. However, there is a simpler method if working with vacuum cleaners or simple robots covering the ground by traversal, instead of just viewing it with a camera. We can lay a grid over the environment. Any vertex of that grid then covered by an obstacle can be deleted from the grid graph, thus forming a grid over the passable terrain. For simple obstacles this leads to the graphs discussed in Section 2.1 to Section 2.3. For the general case of non-rectangular obstacles a planar graph can be used to model the environment. For this case we discuss in Chapter 2 also the exploration of trees.

While graphs represent a discretization of space, a round-based model does the same discretization for time. For the grid graph, time discretization is natural as all distances between neighboring nodes are uniform. For a triangulated graph, this is not necessarily the case.

The capability of a robot to perfectly localize itself in the real world in combination with a reasonable amount of storage space, can be seen in the notion of labeling the nodes in the graph. This allows for algorithms that do not have to do extra bookkeeping tricks like Bender et al. in [Ben94, BFR$^+$02]. Combining these two abstractions allows us to simulate and argue without first needing to solve the simultaneous localization and mapping problem, as we would have to do in a real world testbed using real robots.


(a) A landscape modeled with polygon shapes


(b) Same landscape as grid

Trees Trees are chosen over planar graphs as they represent a hard sub-problem to the online-exploration of general graphs. Meaning we are unable to show a stronger lower bound for general graphs or planar graphs. It should be noted that any graph can be explored with a tree exploration algorithm by building a spanning tree of the graph while exploring it, e.g. newly discovered edges that lead to circles can be ignored. But any guarantee for the run-time of a tree exploration algorithm only refers to this spanning tree created by the exploration algorithm and not the diameter of the graph. It depends on the tree exploration algorithm itself, if an additional adaption from tree to graphs is necessary to prevent extreme spanning diameters from occurring. For example a tree algorithm adapted to the graph could choose to switch its spanning tree or parts of its spanning tree with a shortest path tree, when ever this change halves the depth of the tree it works on or the algorithm is in a position where the change is for free.

After formalizing this model in the next section we will be able to look at problems large groups of robots face:
How to organize efficiently an exploration? (Chapter 2)
How to repeatedly cover a whole graph with robots? (Chapter 3)

1.1 Model

Our algorithms work on an undirected, connected graph $G = (V, E)$ with $|V| = n$. We use d as the depth of a tree. For grids we use $m \times m'$ as notation for a rectangular 2-dimensional grid of which V is a subset. The means the nodes of V are then described as tuples: $V \subseteq [0, \ldots, m-1] \times [0, \ldots, m'-1]$, where any edge $\{(i,j),(i+1,j)\}$ exists if $(i,j),(i+1,j) \in V$ and any edge $\{(i,j),(i,j+1)\}$ exists if the nodes $(i,j),(i,j+1) \in V$. Without restriction we assume $m' \leq m$.

The term **distance** refers to the number of edges on a shortest path between two nodes.

All nodes are labeled with a unique identifier in order to be distinguishable.

The goal of an algorithm $\mathcal{A}$ is to visit any node as soon as possible and then revisit all nodes in as short as possible intervals. To achieve this an algorithm is given k robots that each is assigned to a node of the graph. We require for our algorithms that k is a small polynomial in the graph size: $k = n^c, 0 < c < 1$.

In each round t an algorithm has to decide for each of its k robots where to move. A robot can move from its current node to any node that is adjacent in the graph. There is no restriction made to the time needed to compute the next step for all robots.

The **first visit time of a node** or **exploration time of a node** $t_f(\mathcal{A}, v)$ is the number of the round, in which a robot visits this node for the first time. The **visit time of a node** $t_\theta(\mathcal{A}, v)$ is the supremum of all time intervals between any two visits of this node (revisit) including the time interval necessary for the first visit by any robot. The **long term visit time of a node** is the supremum of time intervals between any two visits of a node by any robot after an arbitrarily long time. The corresponding definitions for the full graph is given by the maximum first visit, visit and long term visit time of all nodes. We will refer to the first visit time for the full graph as $t_f(\mathcal{A})$ and for visit time as $t_\theta(\mathcal{A})$. To shorten notation we will often omit $\mathcal{A}$ and just write t_f or t_θ.

We are analyzing two types of restricted algorithms which we compare to an optimal offline algorithm.

- *Exploration algorithms* are restricted in their knowledge of the graph. They can only use the induced subgraph defined by the already visited nodes of V and their adjacent nodes to compute the next move. This implicitly defines an global communication model, because a global algorithm controlling all robots with full knowledge of the team and all nodes the team explores is equivalent to running a distributed algorithm, i.e. k instances of $\mathcal{A}$ are executed, with full exchange of information between instances. For exploration all robots start assigned to the starting node $s \in V$. We focus with exploration algorithms on minimizing the exploration time t_f, as after the graph is explored the exploration algorithm has no longer any restrictions and can mimic the optimal offline algorithm.
- *Parallel Unaware cleaning algorithms* are restricted in their knowledge about other robots. As an additional challenge cleaning algorithms do not necessarily start with all robots at the same node. All k robots are positioned in the graph on their starting nodes $S = \{s_1, \ldots, s_k\}$. The algorithm has to decide for each robot r which edge to traverse without knowledge of the other robots, basing all its decisions solely on its starting position s_r and the structure of the graph G . This simulates the algorithm is running in a robot unaware of other robots. Robots never learn in these scenarios about the first visit and visit times of nodes, as revealing these would immediately tell the robot the positions of other robots and allow for more effective algorithms. Note that the value r is not provided to the algorithm. If r was known the algorithm would have knowledge that at least r robots are present and additionally know that it is currently controlling the r-th robot.

- *Offline algorithms* are used as baseline algorithm to measure against. These are unrestricted in their knowledge of the graph and aware of all robots. We measure the other algorithms by comparing the ratio of their first visit and visit time with the fist visit time and visit time of the offline algorithm.

$$\textsc{Competitiveness first visit} = \frac{t_f(\mathcal{A}_{\text{Online}})}{t_f(\mathcal{A}_{\text{Offline}})}$$

$$\textsc{Competitiveness visit} = \frac{t_\theta(\mathcal{A}_{\text{Online}})}{t_\theta(\mathcal{A}_{\text{Offline}})}$$

  This analysis of restricted algorithms is called competitive analysis. The restricted algorithms are in competitive analysis synonymously called *online algorithms.* To get a worst case competitive analysis of an algorithm, an adversary, in full knowledge of $\mathcal{A}_{\text{Online}}$, may now choose G,k and S to force the restricted algorithm into its worst case competitive ratio.

Implicitly the parallel unaware cleaning an exploration algorithms define a communication model. For the exploration a single algorithm knowing the

For randomized algorithms, we will use the term *with high probability* to refer to an event which occurs with probability $p = 1 - n^{-c}$ with constant $c \geq 1$. In all of our results, this constant c can be arbitrarily increased if one allows a larger constant factor for the run-time. It is important that the first visit and visit of randomized algorithms is only ever shown with high probability and becomes weaker if we ask for a probability of $p = 1$ instead as we do for the deterministic algorithms.

1.2 Related Work

Exploration using a single robot has been studied for decades (for a survey we recommend [RKSI93]) and can be considered as the online variant of the Traveling Salesman Problem (TSP) [Kar72]. The standard offline version TSP can be considered solved for the euclidean version of TSP, as Arora Sanjeev has shown a polynomial-time approximation scheme(PTAS) [Aro96] for it.

But in contrast to TSP exploration algorithms do not know the graph beforehand and have to find an as cheap as possible tour through a graph that is uncovered with every step. This leads to a totally different problem from the offline version.

For the single robot case, asymptotically optimal exploration up to a factor of two is possible with depth-first search(DFS). Using a map, the exploration of a line or tree can be improved by preventing double traversal of edges. Desmark et al. [DP04] show various competitive constants that can be gained depending whether an anchored, unanchored or no map at all is available. A more abstract concept for a map is used by Fraigniaud et al. in [FIP08]. They show that $\mathcal{O}(\log\log(d))$ bits of information can be enough to explore a tree faster than normal DFS.

Our graphs are labeled to avoid introducing any complexity into the exploration by robots not knowing where they are. If graphs are not labeled, DFS cannot be directly used. M.A. Bender presents solutions for this scenario in [Ben94, BFR+02] using a pebble or a second robot for bookkeeping.

In 2004, Fraigniaud et al. consider the multi-robot exploration problem for trees (later published in journal form in [FGKP06]). They present an algorithm, that with a competitive factor of $\mathcal{O}(k/\log k)$, is far apart from their lower bound of $\Omega(2+\frac{1}{k})$ and quite close to the trivial upper bound of $\mathcal{O}(k)$ achieved by executing a DFS using a single robot. While the lower bound is improved by Dynia et al. in [DLS07] to $\Omega(\frac{\log k}{\log\log k})$, the upper bound remained the state of the art for 10 years until our recursive approach was published in 2014 (O. et al in [OS14]).

Several restrictions for the exploration can improve the bounds. If algorithms are restricted to greedy exploration, an even stronger bound of $\Omega(k/\log k)$ is shown by Higashikawa et al. [HKLT12]. This matches the algorithm of Fraigniaud et al., which is such a greedy algorithm.

For restricted graphs several better algorithms exist. Dynia et al. showed in [DKHS06] a faster exploration for trees restricted by a *density* parameter p, enforcing a minimum depth for any subtree depending on its size. For example, trees embeddable in p-dimensional grids could be explored with competitiveness of $\mathcal{O}(d^{1-1/p})$.

In Brass et al.[BCMGX11] an upper bound of $\mathcal{O}(\frac{n}{k}+d^{k-1})$ is shown. They implement an algorithm that moves robots similar to the method of Fraigniaud et al. [FGKP06], but also works on graphs using only a local communication model with bookkeeping devices.

Dereniowski et al. discuss in their work very large values of k. They show how many more robots need to be invested to explore in asymptotically optimal time. They show a minimum of $k = dn^{1+\epsilon}$ for a constant $\epsilon > 0$ robots to be necessary. This improves the trivial bound of $\mathcal{O}(n^d)$ required to explore any graph in exactly time d by flooding [DDK+13]. They also show their algorithm to be transferable from trees to graphs without loosing the $\mathcal{O}(\frac{1}{\epsilon})$ competitiveness.

Exploration of directed graphs is not discussed here, as graphs are undirected to represent an exploration of normal terrain where you can take a step back if needed. Competitive analysis done by Albers et al. [AH00], Fleischer et al. [FT05], Papadimitriou et al. [DP90] and Förster et al. [FW12] indicates this to be a harder problem than the undirected case. The direction property of the edges allows the adversary to generate sub-graphs that force a robot to redo the whole exploration to get back to a new unexplored node.

Some works model the exploration geometrically, this is useful if robots have a sense of sight enabling to see additional nodes before visiting them [GR03, KKMZ10] or having to move around corners to make everything visible in case of unlimited vision [AKS99, AKS08, CLP11].

We will later take a look at a 2-dimensional geometric model with rectangular obstacles. Papadimitriou et al. discuss with a similar model a online version of the *shortest path problem*([Dij59]). Their goal is to navigate a room instead of exploring it fully. For reaching a line in distance m they show a competitive factor of $\mathcal{O}(\sqrt{m})$ [PY91]. With randomization this bound was later improved to $\mathcal{O}(m^{4/9} \log m)$ by Berman et al. [BBF+96]. Reaching not a line in distance m, but a point in a $m \times m$-grid can be done faster. Bar-Eli et al. show in [BEBFY94] that reaching a given point in time $\mathcal{O}(m \log m)$ is possible. We will use this result as base algorithm for exploring such grids faster than it is currently possible for trees. By itself this result is already enough to explore a $m \times m$-grid graph with $k = m^2$ robots in $\mathcal{O}(m \log m)$-steps. This can be done by navigating each robot to a different vertex in the grid.

A discussion of a 1-dimensional geometric model without obstacles, but offering different speeds for walking and working is done by Czyzowicz et al. in [CGG+15]. They show a 2-competitive algorithm for searching a segment of unknown length with such a heterogeneous team of robots.

Very different results are generated by energy models. While the task stays the same, visiting all nodes in the graph, time is no longer a limit, but the distance traveled by robots. The goal in this model is to minimize the maximum distance traveled by robots, as for example Dynia et al show in [DKS06] to explore trees 8-competitive in the maximum energy needed per robot along with a lower bound of 1.5-competitiveness for any deterministic online-algorithm. Another variant of the energy model is the exploration assuming a fueling station at a starting node, this is called piecemeal exploration, as in Duncan et al. [DKK06]. One more energy-like variant was presented by Das et al. [DDK15]. They assume a fixed energy for the robots and minimize the amount of robots needed for the exploration.

When the graph is known beforehand, as with the parallel unaware cleaning algorithms, the problem resembles more the traveling salesman problem than exploration. Solving TSP with multiple agents is called Multiple Traveling Salesman Problem (mTSP) and is discussed by Bektas et al. [Bek06] and Fredericksen et al. [FHK76].

The mTSP tries to cover the graph with a set of tours and minimizes the length of the longest tour. This corresponds to the offline parallel cleaning problem, if we use the distance between nodes in the graph as cost measure between nodes in mTSP. Even if salesmen start at different nodes, the problem can still be reduced to the regular mTSP [Guo95].

A variant of mTSP that requires robots to repeatedly visit the same area as good as possible is Patrolling algorithms [PR11b]. Robots have to repeatedly patrol along a landscape modeled by a graph or geometrically to detect intruders into an area. Common strategies for this are either performing a tour along a circle on the graph and distributing robots evenly or partitioning the graph and asserting one robot per partition. Recently, this topic has garnered the interest of more research in the field of robotics [ARS+04, Che04, EAK09].

We consider our parallel unaware cleaning problem as a very restricted version of the patrolling problem. Usually patrolling algorithms are allowed communication or robots are able to sense each other for coordination, while we have to work without this coordination. In these works *idleness* of nodes is minimized or *point visit frequency* is maximized which both translate to our *visit time*.

Machado et al [MRZD03] compared in their work several patrolling algorithms with different restrictions empirically. Their category of communication-less algorithms nearly matches our scenario and are therefore worth comparing to. One of those algorithms, the *Conscientious Reactive*, will later be compared to our solution to the problem. Portugal et al. in [PR11a] feature more empirical analysis also including the *Conscientious Reactive Algorithm*.

A similar definition to first visit time is the notion of *cover time* for random walks. Likewise visit time can be compared to the *hitting time* $H(i, j)$, the expected time starting from node i to reach node j. Our robots are not forced to use random walks. Thus, the Lollipop graph, a lower bound construction for the cover time of $\Omega(n^3)$ [Lov96] and obtained by joining a complete graph to a path graph with a bridge (see Figure 1.1), can be cleaned quite efficiently by parallel unaware cleaners.

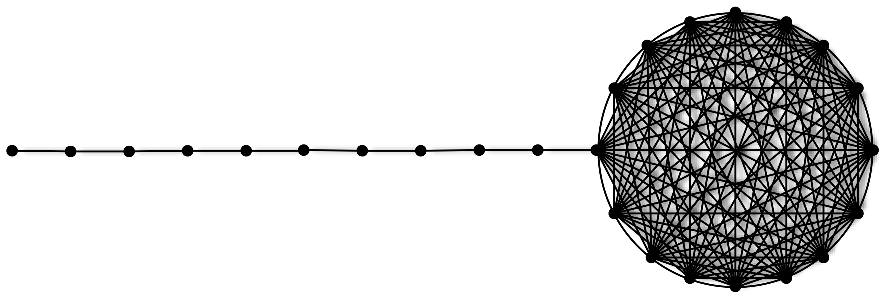


Figure 1.1: Lollipop graph example

1.3 Our Contribution

Most of the work in this thesis has been published previously in conference proceedings.

Chapter 2 about exploration algorithms relies heavily on [OS12], where we have studied exploration of a seemingly simpler subclass of graphs called grid graphs with rectangular obstacles.

We show the lower bound for the exploration of grid graphs to be the same as for trees ($\Omega(\log k/ \log\log k)$) and follow up by discussing how the lower bounds for trees and grid graphs change, if we allow for randomization in the exploration algorithms. It turns out that this does not change the bound for trees, but for the grid graph we are merely able to show a weaker bound of $\Omega(\sqrt{\log m}/ \log\log m)$.

We proceed in Section 2.3 by showing an exploration algorithm for the grid graph which is $\mathcal{O}(\log^2 n)$-competitive. Thereby improving the best bound of $\mathcal{O}(n^{\frac{1}{2}})$ which could be achieved by adapting Dynia et al.'s algorithm from [DKHS06] to the grid.

In the same chapter we present our main result from [OS14], the first online exploration with sub-polynomial overhead of $k^{o(1)}$ on trees with d, k being polynomial in n, or $2^{(2+o(1))\sqrt{(\log d)(\log\log k)}}(\log k)(\log k + \log n)$ for any values of k and d.

Chapter 3 is based upon our work published in [OS15] studying the necessity for communication in an offline exploration like scenario, the parallel unaware cleaning model.

2 Exploration

In this chapter we take a look at the exploration of two interesting graph classes: Trees and grid graphs with convex obstacles. The trees we deem interesting for being the hardest subclass of general graphs known to us. The grid graphs with convex obstacles on the other hand are a class that could be considered a more natural model for environments build by humans. In the following sections we will see lower bounds for both classes and the state of the art exploration algorithms to explore these graph classes.

First, we take a look at the offline algorithms for exploration.

Creating an optimal path P_{opt} through a known graph for a single robot is NP-hard. Ideally we would never touch any node twice which would result in a Hamiltonian path for the given graph. Using k robot changes the problem to finding paths $P_1, \dots,\ P_k$ where $\bigcup_{i=1}^{k} V(P_i) = V(G)$ and we have the goal to minimize the longest path: $t_f = \max_{i=1,\dots,k}\{|P_i|\}$.

There are obvious limits to minimizing the longest path. These limits help us to to provide a simple lower bound an offline algorithm can reach. Each node must be visited and each robot can visit at most one new node each round. The starting node is already visited in the beginning. In addition the node farthest away is in distance d and must be included in at least one path. Therefore we get a lower bound for any offline algorithm of $t_f \geq \max\{\lceil\frac{n-1}{k}\rceil, d\}$. While this is not directly helpful, we can now create an offline algorithm that solves the exploration within a constant factor of this offline lower bound. This is done by using the DFS path over the graph and separate it into k segments, each to be explored by its own robot (See Algorithm 1). This establishes a constant approximation factor of three of this offline solution to any optimal offline solution.

A constant factor offline approximation. A very basic observation is the constant factor offline approximation presented in Algorithm 1, which establishes a constant approximation factor of three.

Algorithm 1: Offline 3-competitive multi-robot exploration of trees for robot r

```
1: Compute a cycle of length $2n-2$ using DFS covering $G$
2: Divide the cycle into $k$ paths of size at most $|P_i| \leq \lceil\frac{2n-2}{k}\rceil$
3: Go to the first vertex of $P_r$
4: Traverse $P_r$
```

Lemma 1 *Algorithm 1 needs at most* $d+\lceil(2n-2)/k\rceil$ *robot moves and has a competitive factor of three.*

Proof: Every exploration algorithm needs at least $\max\{\lceil(n-1)/k\rceil, d\}$ steps. The number of robot moves of Algorithm 1 is at most $t_f \leq d + \lceil(2n-2)/k\rceil \leq 1 \cdot d + 2 \cdot \lceil(n-1)/k\rceil \leq 3 \cdot \max\{d, \lceil(n-1)/k\rceil\}$. Hence, it is 3-competitive. $\square$

In the following sections we will see how well exploration algorithms can at best perform against this offline bound and how good the best algorithms do perform against it.

2.1 Lower Bounds for Exploration

The best lower bound construction for trees was found by Dynia et al. in [DLS07] and called the Jellyfish-Tree in Figure 2.4. The Jelly Fish construction uses $t \geq k$ long path graphs to separate *poisons*, carefully sized conglomerate of nodes, to force any online algorithm to redistribute its robots $\Omega(\frac{\log k}{\log\log k})$ times. In this section we show a construction inspired by this work for grid graphs with only rectangular obstacles.

Since we take a look at the asymptotic behavior, we restrict values for the side length m of the overall grid and the number of robots k to be powers of 2. For our construction we choose $m = k^2$. We separate G into k rectangular sub-grids, our poisons, such that the shortest path between every pair of these areas has a length of $\Theta(m)$, see Figure 2.1.

Given the paths of robots in a poison area visiting at most w nodes in total and each robot starting (or departing) at one of the four corners of an area of $m/2 \times m/k$ we construct the grid poison in the following way: For each $j \in \{1, \ldots, \log m - \log k\}$ consider a sub-grid of $2^j \times 2^j$-squares. If no robot has visited a square, then all vertices of this square except the border nodes (leftmost and rightmost column, lowermost uppermost row) will be removed. The border nodes are necessary to ensure that the square shaped obstacles remain disjoint and that the robot in the neighboring square does not learn anything. We call w the fooling size of the grid poison.

Lemma 2 *Given a deterministic strategy of robots where the number of all traversed nodes of the robots is at most* w *in a* $m \times m'$ *(with* $m \geq m'$*) rectangle, then the corresponding poison has at most size* $\mathcal{O}(m + w\log m')$*. No visited vertex is adjacent to a rectangle.*

Proof: Clearly, the w traversed vertices remain in the graph. When a sub-grid of dimension $2^j \times 2^j$ is removed, then $2^{j+2} - 4$ vertices remain in the graph.

We estimate the number of such sub-grids which can be reached by any robots. Four sub-grids can be reached without any traversal since the robots may start at the corners. The explored sub-grids are connected since they result from a set of paths starting in the corners. So, at most $4 + w2^{2-j}$ sub-grids of dimension $2^j \times 2^j$ can be reached.

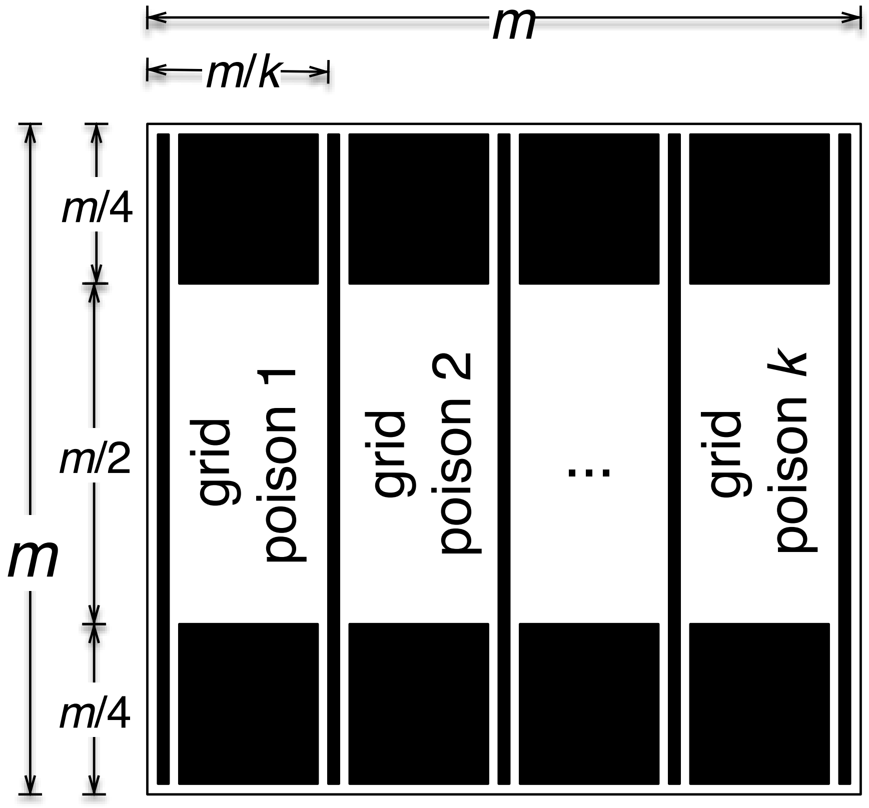


Figure 2.1: Separation of the grid poison areas

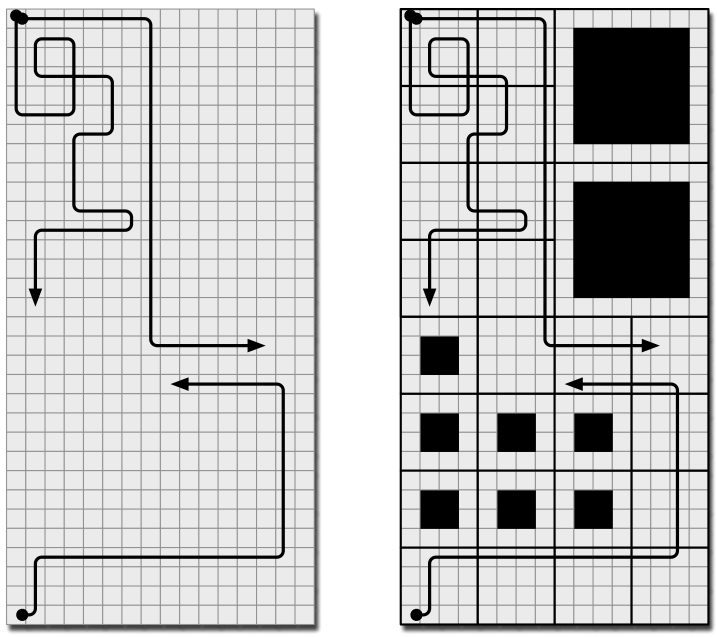


Figure 2.2: Deterministic construction of the grid poison based on the exploration paths of robots

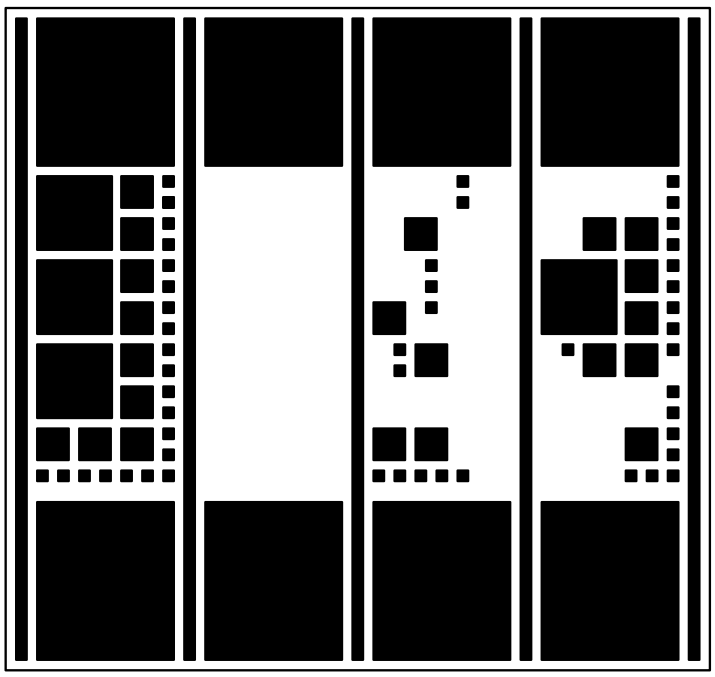


Figure 2.3: Example graph for the deterministic lower bound

Note: This is a slight overestimation as we can actually only reach two and not four new sub-grids by traversing the edge length of one sub-grid.

We want to count all $2^j \times 2^j$ sub-grids which are replaced by an obstacle. Each such sub-grid has three neighbor sub-grids (horizontal, vertical and diagonal) in its superordinate $2^{j+1} \times 2^{j+1}$ sub-grid, of which at least one has been visited by a robot. Otherwise, the superordinate sub-grid would have been replaced by an obstacle. So the number of replaced sub-grids is at most $4(4+w2^{2-j})$. In each of the $m' \times m'$ sub-squares of the $m \times m'$ grid poison with at most w_i robot paths we observe at most the following number of vertices.

$$\sum_{j=2}^{\log m'} 4(4 + w_i 2^{2-j}) \cdot (2^{j+2} - 4) \leq 64m' + 64 w_i \log m'$$

Summing over all such m/m' squares we get at most
$64m + 64w \log m' = \mathcal{O}(m + w \log m')$ vertices. □

While w visited vertices are not enough to encounter any obstacle, we show that visiting $\mathcal{O}(w \log m' + m)$ vertices suffice to visit all vertices. An offline strategy with at most $k \leq m$ robots can do so in time $\mathcal{O}(\frac{w}{k} \log m' + m)$.

Lemma 3 *A grid poison with fooling size w in a $m \times m'$ rectangle ($m \geq m'$) can be explored by k' robots in time $\mathcal{O}(m + \frac{w}{k'} \log m')$ in the offline setting.*

Proof:

Partition the $p \leq 64m + 64w \log m'$ vertices of the grid poison in $b_i \times m'$ rectangles such that $\sum_{i=1}^{k'} b_i = m$ and that the number of vertices in each rectangle is at most $\frac{p}{k'} + m'$. Each of the k' robot explores one such rectangle. It needs m steps to reach the rectangle. For exploring such a rectangle, a robot may have to take a detour into neighbored rectangles because an obstacle hinders the direct path. Such detours have at most $4m'$ vertices. Furthermore, paths inside the rectangle may be traversed at most twice. This leads to an upper bound for $2b_i + 4m'$ for the exploration within the rectangle. To reach the rectangle at most m steps are necessary. This results in an exploration time of at most $m + 6m' + 2\frac{p}{k} = \mathcal{O}(m + \frac{w \log m'}{k'})$. □

The fooling size of the grid poisons is chosen according to the following distribution:

$$w_{\sigma(i)} = \left\lceil \frac{km}{(\log k)^2} \cdot \frac{1}{i} \right\rceil$$

Where σ denotes a permutation depending on the deterministic exploration strategy.

Lemma 4 *There is an offline strategy which explores this graph within $\mathcal{O}(m)$ steps using k robots.*

Proof: Remember that $m = k^2$ and $m' = k$. Define $\mathcal{W} := \sum_{i=1}^{k} w_{\sigma(i)}$ and note that $\mathcal{W} = \mathcal{O}\left(k + \frac{km}{\log k}\right)$. An offline exploration strategy sends one robot in each grid poison for time cm to explore the grid poison. After this round, it sends $\left\lfloor \frac{w_{\sigma(i)}}{\mathcal{W}} k \right\rfloor$ robots in each unexplored grid poison for time $c \cdot m$ as well.

All poisons with fooling size of at most $m/\log k$ can be explored within the first round. If $w_{\sigma(i)} > \frac{\mathcal{W}}{k}$, then at least one robot will explore the grid poison after this round. This is the case for $i \leq c\frac{k}{\log k}$ for some constant $c > 0$. Exploring such a poison costs time linear in

$$m + \frac{w_{\sigma(i)}}{\left\lfloor k \frac{w_{\sigma(i)}}{\mathcal{W}} \right\rfloor} \log m' = \mathcal{O}\left(m + \frac{\mathcal{W}}{k} \log m'\right) = \mathcal{O}(m) \ .$$

□

Theorem 1 *Any deterministic exploration strategy needs at least* $\Omega\left(m \cdot \frac{\log k}{\log \log k}\right)$ *steps to explore this graph with* k *robots.*

A proof is analogous to the lower bound in [DLS07]:

Proof: For the lower bound argument we consider rounds of length $m/2$. Note that in each round a robot can visit only one poison grid. Let $k_{t,i}$ denote the number of robots that visit poison T_i in round t. At the beginning of each round the adversary allows the robots to know the size of some of the poisons while a decreasing number of poisons remain of unknown size. As soon the robots learn the size of the poison the poison is lost and no more exploration costs are accounted for (since the overall offline exploration cost is $\mathcal{O}(m)$). Furthermore, we do not count the costs of replacement from one poison grid to another.

Let u_t be the number of unexplored poisons after the t-th round. Then, at least $\frac{u_t}{2}$ poisons are explored by at most $2\frac{k}{u_t}$ robots. By our construction we can ensure that

$$u_t \geq \frac{k}{2^t \log^{2t} k} \ .$$

For this we adapt the permutation $\sigma_{\mathcal{A}}$ (we will write σ for brevity) to an exploring Algorithm $\mathcal{A}$ in the following way. All the poisons are sorted according to whether $\mathcal{A}$ sends $k_{t,i} \geq 2\frac{k}{u_t}$ robots into a poison in the t-th round. For this purpose we place the poison grids with larger number of robots in the first round at the beginning, then we continue within the set at the beginning by sorting poison grids according to the robots of the second round and so forth.

By induction before round t at least u_{t-1} poisons are unexplored. Now the search algorithm can place at most k robots among those poisons and at least $\frac{u_{t-1}}{2}$ poisons are explored by at most $2\frac{k}{u_{t-1}}$ robots in round t. Hence, the robots cannot explore those poisons where

$$|T_{\sigma(i)}| \geq m2^t \log^{2t-2} k \geq km\left(\frac{1}{u_{t-1}} + \frac{1}{u_{t-2}} + \ldots + \frac{1}{u_0}\right)$$

(by induction) since the robots have only time $m/2$ to explore the poison area. The number of poisons u_t of this size can be evaluated by using the definition of the distribution.

$$\frac{km}{u_t(\log k)^2} \leq m2^t \log^{2t-2} k$$

Thus, we have

$$u_t \geq \frac{k}{2^t \log^{2t} k}$$

which proves the number of unexplored poisons by induction.

Note that for $t_{last} = \frac{1}{4}\frac{\log k}{\log\log k}$ we have

$$u_{t_{last}} = \frac{k}{2^{2t_{last}} \log^{2t_{last}} k} \geq \frac{k}{k^{\frac{1}{2\log\log k}} 2^{\frac{1}{2}\log k}} \geq \frac{k}{k^{\frac{1}{2}} k^{\frac{1}{2}}} = 1 \ .$$

By this construction, we have at least $\Omega\left(\frac{\log k}{\log\log k}\right)$ rounds with unexplored poison grids where each of the rounds have a run-time of $\frac{m}{2}$. $\square$

This implies an exploration time for any deterministic online algorithm of

$$\Omega\left(m \cdot \frac{\log k}{\log\log k}\right)$$

leading directly to a lower bound for the competitive ratio of

$$\Omega\left(\frac{\log k}{\log\log k}\right)$$

Therefore, we achieve the same lower bound for grid graphs with rectangular obstacles that was achieved for trees. In section 2.3 we will show a exploration algorithm that nearly matches this bound.

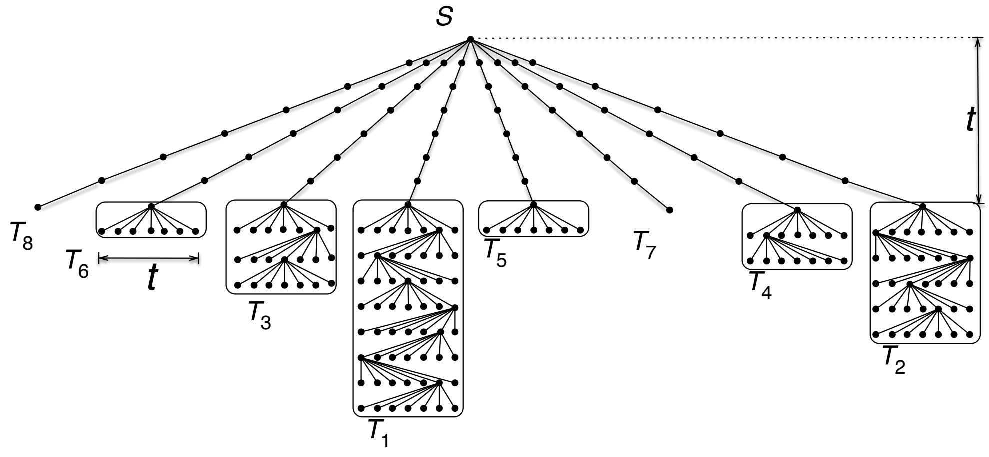


Figure 2.4: Jellyfish-Tree construction from [DLS07].

2.2 Lower Bounds for Randomized Algorithms

As in the last section very strict knowledge of the movement of an algorithm was used to create poisons, we now take a look at randomized algorithms. We will show in this chapter that while randomization makes it harder to create poisons and show lower bounds, it does not prevent the construction of poisons completely. We show in this section that a similar bound for trees and a slightly weaker bound for the exploration of grid graphs with rectangular obstacles exists than it does for deterministic algorithms.

Lower Bound for Randomized Tree Exploration

We first show that the lower bound given in [DLS07] for trees also applies for randomized algorithms. Consider the Jellyfish-Tree in Figure 2.4. We use the same construction with k subtrees and a random permutation σ over $\{1, \ldots, k\}$. The i-th subtree consists of a path of length $t = k$ and a poison which is a tree of size $|T_{\sigma(i)}|$ and depth $\frac{|T_{\sigma(i)}|}{t}$ where

$$|T_{\sigma(i)}| := \left\lceil \frac{k}{\log k} \cdot \frac{1}{i} \right\rceil \cdot t$$

where in each level $t-1$ leaves are connected to a parent and the graph continues at a random child, which we call the target child. The permutation σ is chosen uniformly at random. In [DLS07] the following lemma has been shown regarding the offline exploration time.

Lemma 5 *The Jellyfish-Tree can be explored in time $\mathcal{O}(t)$ using k robots.*

Yao's principle [Yao77] is used to show a lower bound for randomized strategies. We choose the randomized Jellyfish-Tree for a deterministic exploration strategy and show a lower bound on the expected time.

Theorem 2 *For every randomized online exploration algorithm $\mathcal{A}$, there is a graph such that $t_f(\mathcal{A})$ is at least $\Omega\left(\frac{\log k}{\log\log k}\right)$ times larger than the optimal time needed to explore this graph offline by k robots.*

Proof: We consider rounds of length t. In each round a robot can visit only one poison. We assume that the deterministic exploration strategy knows the graph family. Therefore it has determined a poison, if it has found all target children.

Now in each round of length t steps a different number of robots might explore a poison.

Lemma 6 *The probability that in a round a target child in depth ℓ is explored with $k' \leq t$ robots in less time than $\frac{1}{2}\frac{t}{k'}$ is at most $e^{-\frac{1}{8}\ell}$.*

Proof: We assume that all k' robots test different children where each child is a target child with equal probability. Then, the probability to find the target child of the next level within i steps is $i \cdot \frac{k'}{t}$.

Define the random variable X which denotes the number of steps to find one target child with k' robots. Then $P[X = j] = \frac{k'}{t}$ for $j \in \{1, \ldots, \lceil t/k' \rceil - 1\}$ and $P[X = \lceil t/k' \rceil] = \frac{t - k' \bmod t}{t}$. Clearly, $\frac{1}{2}\lfloor t/k' \rfloor \leq E[X] \leq \frac{1}{2}t/k'$.

If $k' \geq t/2$ the target child may be found in each step. Otherwise if $k' < t/2$ we can bound the number of steps to find a series of ℓ target children using Hoeffding's inequality from [Hoe63a] (Theorem 2). Assume ℓ independent target children and let $\mathcal{S}_\ell = \sum_{j=1}^{\ell} X_{j,i}$, where $X_{j,i}$ denotes the random variable above for k' robots. Then, by the tail inequality, we have for all $t \geq 0$ and $a_i = 1$ and $b_i = \lceil t/k' \rceil$

$$P[\mathcal{S}_\ell - E[\mathcal{S}_\ell] \leq -\delta] \leq e^{-2\delta^2 / \sum_{i=1}^{\ell}(b_i - a_i)^2}$$

Since $b_i - a_i \leq t/k'$ we have

$$P\left[\mathcal{S}_\ell - E[\mathcal{S}_\ell] \leq -\delta\right] \leq e^{-2\delta^2 k'^2/(\ell t^2)}$$

We choose $\delta = \frac{1}{2}E[\mathcal{S}_\ell] \geq \frac{1}{2}\ell \cdot \lfloor \frac{t}{k'} \rfloor$ and get for $k' \leq t/2$

$$\begin{aligned} P\left[\mathcal{S}_\ell \leq \frac{1}{2}E[\mathcal{S}_\ell]\right] &\leq e^{-\frac{1}{2}\ell^2 \lfloor t/k' \rfloor^2 k'^2/(\ell t^2)} \\ &\leq e^{-\frac{1}{2}\ell(1-k'/t)^2} \leq e^{-\frac{1}{8}\ell} \end{aligned}$$

Remember that for $k' > t/2$ we have

$$P\left[\mathcal{S}_\ell \leq \frac{1}{2}E[\mathcal{S}_\ell]\right] = 0$$

The probability that a target child in depth ℓ is explored with k' robots in less time than $\frac{1}{2}\frac{t}{k'}$ is at most $e^{-\frac{1}{8}\ell}$. $\square$

This implies the following corollary which shows that with high probability $1-\frac{1}{n^2}$ that maximum speedup by randomization in a poison is a factor of $\mathcal{O}(\log n)$.

Corollary 1 *The probability that in a round of length t a target child in depth $16\ln n$ is explored with $k' \leq t = k$ robots in less time than $\frac{1}{2}\frac{t}{k'}$ is at most $\frac{1}{n^2}$.*

So, k' robots in a round of length t can only find all target children in depth of at most $32k'\ln n = (64\ln 2)k'\log k$ with probability $1-\frac{1}{n^2}$.

Now, in the first round, we have k robots which are (deterministically) each assigned to a poison, but necessarily each poisons receives at least one robot. They have at most time t to explore each poison.

Consider poisons of depth of at least $c^j \log^j k$ for $c = 2^8$. There are as many as $\frac{k}{c^j \log^{j+1} k}$ such poisons. Such a poison cannot be explored with less than $\frac{c^j \log^{j-2} k}{2^6 \ln 2}$ robots with high probability. Then, only target children up to depth $c^j \log^{1-j} k$, i.e. a fraction of $\frac{1}{\log k}$ of the poison can be explored with high probability.

Correspondingly, there are at most $\frac{2^6(\ln 2)k}{c^j \log^{j-2} k}$ poisons which may have enough robots and these poisons are randomly distributed over the set of all poisons. The probability that more than $\frac{c^j \log^{j-2} k}{2^6 \ln 2}$ robots are assigned to a poison is at most $\frac{2^6(\ln 2)}{c^j \log^{j-2} k}$.

The expected number of explored poisons of this depth $c^j \log^j k$ after the first round is at most

$$r = \frac{k}{c^j \log^{j+1} k}\frac{2^6(\ln 2)}{c^j \log^{j-2} k} = \frac{2^6(\ln 2)k}{c^{2j} \log^{2j-1} k} \ .$$

We can apply a Chernoff bound [MU05] since the success of sending the correct or a higher number of robots into a poison is negatively correlated: The exploration of a poison decreases the probability that another poison is explored. For $r \geq 8\ln n$ we get with high probability that at most $2r$ poisons are explored with high probability which is the case for $2 \leq j \leq \frac{1}{4}\log k/\log\log k$. Further, note that for $j \geq 2$

$$2r \leq \frac{k}{c^{j+1}\log^{j+1} k}$$

since

$$\frac{2^7(\ln 2)k}{c^{2j}\log^{2j-1} k} \leq \frac{k}{c^{j+1}\log^{j+1} k}$$

and

$$c^j \log^j k \geq (2^7 \ln 2)c \log^2 k \ .$$

because $c \geq 2^7 \ln 2$.

Hence, the number of unexplored poisons of depth at least
$c^j \log^j k$ for $j \geq 2$ is at least

$$\left(1 - \frac{1}{c}\right) \frac{k}{c^j \log^{j+1} k}$$

after the first round with high probability.

By induction, at the beginning of the $(u+1)$-th round we have at least

$$(1 - \frac{1}{c})^u k c^{-j} \log^{-j-1} k$$

unexplored poisons of depth at least $c^j \log^j k$ for $j \geq 2u$ and $j \leq \frac{1}{4} \log n / \log\log n$. We assume that these bounds are tight.

Again, for poisons of depth at least $c^j \log^j k$ a number of $\frac{c^j \log^{j-2} k}{2^6 \ln 2}$ of robots is not able to explore more than a fraction of $\frac{1}{\log k}$ of such poisons. So, there at most $\frac{2^6 (\ln 2) k}{c^j \log^{j-2} k}$ poisons which may have enough robots and these poisons are randomly distributed over the set of all unexplored poisons which is at least
$(1 - \frac{1}{c})^u k c^{-2u} \log^{-2u-1} k$. The probability that enough robots are assigned to a poison is therefore at most

$$\begin{aligned} &\frac{2^6 (\ln 2) k}{c^j \log^{j-2} k} \left(\left(1 - \frac{1}{c}\right)^u k c^{-2u} \log^{-2u-1} k \right)^{-1} \\ &\qquad = 2^6 (\ln 2) c^{2u-j} \log^{2u-j+1} k \ . \end{aligned}$$

The expected number of explored poisons is at most

$$\begin{aligned} r &= \frac{\left(1 - \frac{1}{c}\right)^u k}{c^j \log^{j+1} k} \cdot 2^6 (\ln 2) c^{2u-j} \log^{2u-j+1} k \\ &= 2^6 (\ln 2) \left(1 - \frac{1}{c}\right)^u c^{2u-2j} k \log^{2u-2j-2} k \end{aligned}$$

Note that for $j \geq 2u + 2$

$$2r \leq \left(1 - \frac{1}{c}\right)^u \frac{k}{c^{j+1} \log^{j+1} k}$$

since

$$\begin{aligned} &2^7 (\ln 2) \left(1 - \frac{1}{c}\right)^u c^{2u-2j} k \log^{2u-2j-2} k \\ &\qquad \leq \left(1 - \frac{1}{c}\right)^u \frac{k}{c^{j+1} \log^{j+1} k} \end{aligned}$$

and

$$c^{j-2u} \log^{j-2u} k \geq (2^7 \ln 2) c \log^2 k \ .$$

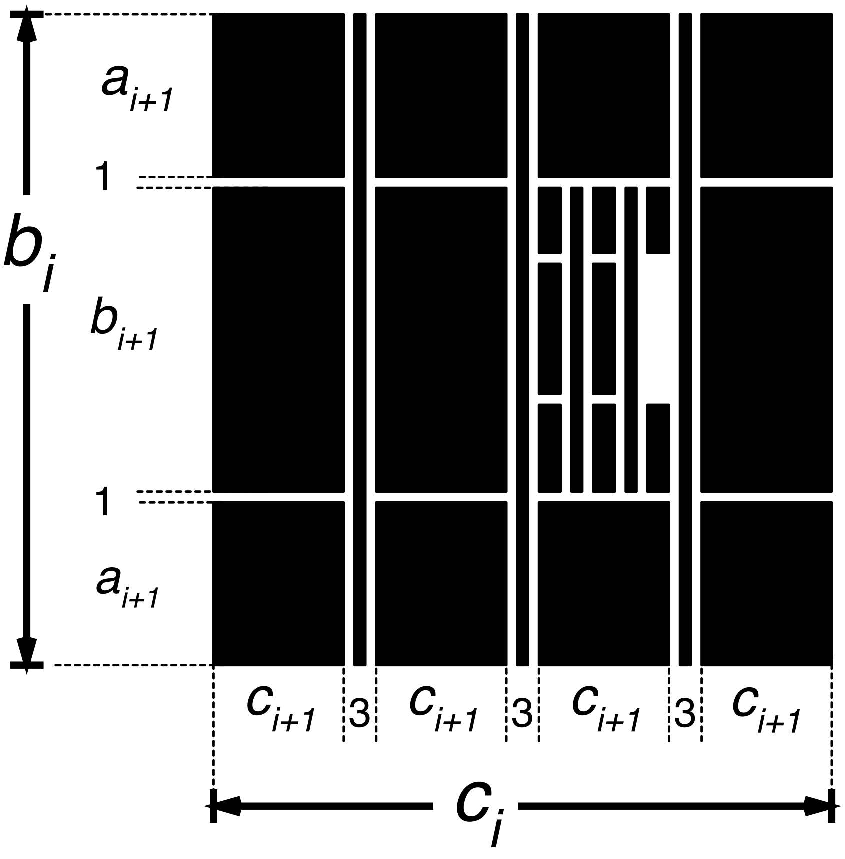


Figure 2.5: Recursive construction of poison areas for the randomized lower bound

because $c \geq 2^7 \ln 2$. Again, we can apply Hoeffding's bound [Hoe63b] since the explored poisons are negatively correlated. Applying Hoeffding's bound for $r \leq 8 \ln n$, we get with high probability that at most $2r$ poisons are explored with high probability, which is the case for $2u \leq j \leq \frac{1}{4} \log k / \log \log k$.

Therefore, the number of unexplored poisons of depth $\geq c^j \log^j k$ for $j \geq 2u + 2$ is at least

$$\left(1 - \frac{1}{c}\right)\left(1 - \frac{1}{c}\right)^u \frac{k}{c^j \log^{j+1} k}$$

after the $(u+1)$-th round with high probability, which proves the induction.

Since for all $u \leq \frac{1}{4} \frac{\log k}{\log \log k}$ we can find unexplored poisons with high probability, the claim follows. □

Lower Bound for Randomized Grid Exploration

This theorem can be transferred to grids with rectangular bounds. Again we use a construction with poison areas as in the deterministic lower bound for the grid. Obviously, we cannot use the same construction, since it heavily depends on the knowledge of the deterministic strategy.

Therefore, we use a randomized recursive construction where we place one poison within another. This "Matryoshka doll"-like construction has two features. First, it is hard to find the next enclosed grid poison. Second and most astonishingly: It is bigger on the inside. Every next Matryoshka doll inside is twice as large as the outer one. This is possible, due to the fact that the outer poison has much fewer (yet longer) paths.

The outer grid construction is depicted in Figure 2.2. We construct poison areas specifically designed for $k = \sqrt{m}$ randomized robots where the overall size of the grid is $m \times m$. The whole area is designed in a way that there is an offline strategy where k robots explore the graph in $\mathcal{O}(m)$ steps.

We recursively define the poison areas as depicted in Figure 2.5 starting with the uppermost layer which fits into the overall construction with $a_0 = \frac{1}{4}m - 2$, $b_0 = b = \frac{1}{2}m$ and $c_0 = c = \frac{m}{k+3}$ in the overall construction of Figure 2.2. Further, define for $i \geq 0$

$$a_{i+1} = \frac{1}{4}b_i - 2 \ , \quad b_{i+1} = \frac{1}{2}b_i \ , \quad c_{i+1} = c_i \frac{1}{2^{2i}}$$

So, we have the closed form for $i \geq 0$:

$$a_i = m\ 2^{-i-2} - 2 \ , \quad b_i = m\ 2^{-i-1} \ , \quad c_i = \frac{m}{k 2^{i(i+1)}}$$

This recursive definition ends when $c_{r+1} \leq 1$ for some r. Therefore

$$\frac{m}{k 2^{(r+1)(r+2)}} \geq 1$$

which is implied by

$$\log m \geq \log k + (r+1)^2$$

And therefore:

$$r \leq \sqrt{\log m - \log k} - 1 = \frac{\sqrt{2}}{2}\sqrt{\log m} - 1$$

Note that the recursive constructions replace at most one of the inner rectangles with the next level. In this construction, one of the inner obstacles is replaced by another element. In the lowest level this obstacle is a barrier. The following lemma describes the length of all paths in the fixed level i of the recursion.

Lemma 7 *For a grid poison of level $i \geq 1$, the complete area to be explored is at most $m2^i$.*

Proof: We have at least $\lceil \frac{c_i}{c_{i+1}+3} \rceil$ vertical paths of length b_i which have an overall length of $b_i = m2^{-i-1}$. Note that by definition

$$\frac{c_i}{c_{i+1}} = 2^{2i}$$

Therefore

$$\frac{c_i}{c_{i+1}+3} b_i \leq 2^{2i}\ m\ 2^{-i-1} = m2^{i-1}$$

The length of all horizontal paths is bounded by $4c_i \leq m$. □

We define the workload of an exploration strategy in a poison as the sum of all paths of all robots.

Lemma 8 *For all $p \in [0,1]$ in a grid poison of level $i \geq 1$ the next recursive grid poison has not been found with a workload of at most $\frac{1}{2}pm2^i$ with probability $1-p$.*

Proof: Consider the paths of a deterministic strategy of length $w = pm2^i$. The expected number of possible poisons that can be inspected with this workload is at most $\frac{1}{2}\frac{pn2^i}{b_i} = \frac{1}{2}\frac{pn2^i}{n2^{-i-1}} = p2^{2i}$. Clearly the probability is p for finding the correct target and therefore $1-p$ for failing to do so. □

Now we choose the levels ℓ_j of the poisons $1, \ldots, k$ according to the following distribution where σ is a random permutation over $\{1, \ldots, k\}$.

$$|T_{\sigma(i)}| = \frac{tk}{\log k} \cdot \frac{1}{i}$$

and

$$\ell_j = \lfloor \log |T_j| \rfloor$$

The maximum size of such a grid poison is bounded by
$t2^{\mathcal{O}(\sqrt{\log k})}$. So, we replace our distribution of poison sizes with

$$|T_{\sigma(i)}| = \begin{cases} \dfrac{tk}{\log k} \cdot \dfrac{1}{i} & \text{if } \frac{k}{2^{\mathcal{O}(\sqrt{\log k})} \log k} \leq i \\ m2^{\mathcal{O}(\sqrt{\log k})} & \text{else} \end{cases}$$

where we round to the next power of two.

Theorem 3 *For every randomized online exploration algorithm, there is a grid graph with disjoint rectangular obstacles such that the exploration time is at least $\Omega\left(\frac{\sqrt{\log k}}{\log \log k}\right)$ times longer than the optimal time needed to explore this graph offline by k robots.*

Proof: The proof is analogous to the proof of the randomized lower bound for graphs. The first difference is that we do not prove with high probability, but with probability $1 - \frac{1}{\log n}$. This probability for each poison is large enough since the expected number of unexplored poisons is considered.

The second difference is that the number of rounds is now limited by $r = \frac{\sqrt{\log k}}{\log \log k}$. This is the reason for the worse lower bound. We consider rounds of length $m/2$. In each round a robot can visit only one poison grid. We use Yao's principle [Yao77]and consider a deterministic strategy on the random graphs.

Consider poison grids of level of at least $\log(c^j \log^j k)$ for $c = 2^8$. There are $\frac{k}{c^j \log^{j+1} k}$ many such poison grids. Not even a fraction of $\frac{1}{\log^2 k}$ of such a poison grid can be explored in a round of length $m/2$ with less than $c^j \log^{j-2} k$ robots with probability $1 - \frac{1}{\log^2 k}$. We can bound the number of poisons that are explored in the error case with Chernoff bounds. If $\frac{k}{c^j \log^{j-2} k} \geq 8 \ln m$, then the error probability that more than $2\frac{k}{c^j \log^{j-2} k}$ such poison grids are explored is at most $\frac{1}{m^2}$.

So, there are at most $3\frac{k}{c^j \log^{j-2} k}$ poison grids which may have enough robots and these poisons are randomly distributed over the set of all poisons w.h.p. The probability that more than $c^j \log^{j-2} k$ robots are assigned to a poison is at most $c^{-j} \log^{2-j} k$.

The expected number of explored poisons of this level $\log(c^j \log^j k)$ after the first round is at most

$$r = \frac{k}{c^j \log^{j+1} k} \frac{3}{c^j \log^{j-2} k} = \frac{3}{c^{2j} \log^{2j-1} k}$$

We can apply a Chernoff bound since the explored poisons are negatively correlated: The exploration of a poison decreases the probability that another poison is explored. For $r \geq 8 \ln m$, we get with high probability that at most $2r$ poisons are explored with high probability which is the case for $2 \leq j \leq \sqrt{\log k}/\log\log k$.

Further, note that for $j \geq 2$

$$2r \leq \frac{k}{c^{j+1} \log^{j+1} k}$$

since

$$\frac{6k}{c^{2j} \log^{2j-1} k} \leq \frac{k}{c^{j+1} \log^{j+1} k}$$

and

$$c^j \log^j k \geq 6c \log^2 k \ .$$

if we choose $c \geq 6$.

Hence, the number of unexplored poison grids of level at least $\log(c^j \log^j k)$ for $j \geq 2$ is at least

$$\left(1 - \frac{1}{c}\right) \frac{k}{c^j \log^{j+1} k}$$

after the first round with high probability.

After each round the robots may be placed on different poison grids. Although the robots need time $m/2$ to travel from one poison grid to another, we do not use this feature, since we also have to deal with robots which do not travel to new poisons which complicates the analysis. It is easy to see that taking the travel time into account accounts only for a constant factor.

By induction, at the beginning of the $(u+1)$-th round we have at least

$$(1-\frac{1}{c})^u k c^{-j} \log^{-j-1} k$$

unexplored poison grids of level at least $\log(c^j \log^j k)$ for $j \geq 2u$ and $j \leq \sqrt{\log m}/\log\log m$. We assume that these bounds are tight, i.e. we allow the robot strategy to learn about the situation in the other poison grids.

Again, for poison grids of level at least $\log(c^j \log^j k)$ a number of $c^j \log^{j-2} k$ of robots is not able to explore more than a fraction of $\frac{1}{\log^2 k}$ of such poisons with probability $1-\frac{1}{\log^2 k}$. Again, we bound the error case by Chernoff bound if $\frac{k}{c^j \log^{j-2} k} \geq 8 \ln m$ then the error probability that more than $2\frac{k}{c^j \log^{j-2} k}$ such poison grids are explored is at most $\frac{1}{m^2}$.

So, there at most $\frac{3k}{c^j \log^{j-2} k}$ poisons which may have enough robots and these poisons are randomly distributed over the set of all unexplored poisons which is at least $(1-\frac{1}{c})^u k c^{-2u} \log^{-2u-1} k$. The probability that enough robots are assigned to a poison is therefore at most

$$\frac{3}{c^j \log^{j-2} k}\left(\left(1-\frac{1}{c}\right)^u k c^{-2u} \log^{-2u-1} k\right)^{-1}$$
$$= 3c^{2u-j} \log^{2u-j+1} k$$

The expected number of explored poisons is at most

$$\begin{aligned} r &= \frac{\left(1-\frac{1}{c}\right)^u k}{c^j \log^{j+1} k} \cdot 3c^{2u-j} \log^{2u-j+1} k \\ &= 3\left(1-\frac{1}{c}\right)^u c^{2u-2j} k \log^{2u-2j-2} k \end{aligned}$$

Note that for $j \geq 2u+2$

$$2r \leq \left(1-\frac{1}{c}\right)^u \frac{k}{c^{j+1} \log^{j+1} k}$$

since

$$6\left(1-\frac{1}{c}\right)^u c^{2u-2j} k \log^{2u-2j-2} k \leq \left(1-\frac{1}{c}\right)^u \frac{k}{c^{j+1} \log^{j+1} k}$$

and

$$c^{j-2u}\log^{j-2u} k \geq 6c\log^2 k\ .$$

because we chose $c \geq 6$.

Again, we can apply Hoeffding's bound since the explored poisons are negatively correlated. Applying Hoeffding's bound for $r \geq 8\ln m$ we get with high probability that at most $2r$ poisons are explored with high probability which is the case for $2u \leq j \leq \sqrt{\log k}/\log\log k$.

Accordingly, the number of unexplored poisons of depth at least $c^j \log^j k$ for $j \geq 2u+2$ is at least

$$\left(1-\frac{1}{c}\right)\left(1-\frac{1}{c}\right)^u \frac{k}{c^j\log^{j+1} k}$$

after the $(u+1)$-th round with high probability, which proves the induction.

So, for $\sqrt{\log m}/\log\log m$ rounds of length m, there will be unexplored poisons with high probability. □

These lower bounds could be generalized to robots with vision where each cell needs only to be seen by the robots (and not necessarily visited). This can be done by placing small view obstructing squares at all junctions in the lower bound construction presented here. However such an argument is, of course of questionable utility, as such a universal counter-strategy for vision defeats the purpose of modeling the line-of-sight at all.

2.3 Exploration with Rectangular Obstacles

In this section we take a look at the exploration of grid graphs with rectangular obstacles. Contrary to normal trees grid graphs allow us a sense of direction or area. This allows us to devise a divide and conquer strategy for multi-robot exploration that is faster than any algorithm working on trees.

Applying Bar Eli et al.'s result from [BEBFY94] for navigating to any point in a $m \times m'$-grid ($m \geq m'$) with unknown oriented rectangular disjoint obstacles in time $\mathcal{O}(m \log m')$ or to the obstacle in which the point lies, we present an efficient divide-and-conquer strategy. We use the following notations:

Let N, E, S, W denote the directions. A $\delta_1\delta_2$-path is a directed path which consists only of steps with directions δ_1 and δ_2. For neighbored directions $\delta_1, \delta_2 \in \{N, E, S, W\}$ a *greedy $\delta_1\delta_2$-path* is a path without obstacles where from the starting point the path goes to direction δ_1. Each time an obstacle occurs, the path takes a turn in direction δ_2 and continues until the way is free again in direction δ_1, then it continues in direction δ_1. From every point in the grid, every greedy $\delta_1\delta_2$-path exists and has a maximum length of $2m - 1$.

We need the notion of *surroundable regions.*

Definition 1 *A* surroundable region *is a set of connected nodes which has a bounding path which is described by the concatenation of a greedy NW, WS, SE, and EN-path. Note that the complete $m \times m'$-grid is such a* surroundable region.

The continuous area $A(R)$ of a region R is the area of the region of R where the bounding path and the region is interpreted geometrically.

Every *surroundable region* has a bordering path with a length of at most $4m-2$. For our divide-and-conquer algorithm we successively partition such regions. We also consider a geometric version of the grid graph in the Euclidean plane bounded to $[0, m-1] \times [0, m'-1]$. Obstacles are obviously modeled by rectangles. For the paths of the robot we consider series of line segments connecting the middle points of the empty squares representing the nodes of the graph.

Lemma 9 *Each* surroundable region *R can be partitioned into two* surroundable regions *R_1, R_2 such that $R_1 \cup R_2 = R$ and $A(R_i) \leq \frac{3}{4}A(R)$ for $i \in \{1, 2\}$. This can be done in time $\mathcal{O}(m \log m)$ with a single robot.*

Proof: For a fixed region R consider a point $p = (p_x, p_y)$ in this Euclidean space. Then we define $Q_1(p)$ as the area of R in the NE-quadrant (including obstacles). Similarly, we define Q_2, Q_3, Q_4 as the areas of R in the NW, SW and SE-quadrant. Clearly, the sum of all $Q_i(R)$ equals the area $A(R)$ of R. See Figure 2.6 for visualization.

Lemma 10 *There exists a point p such that:*

1. *$Q_1(p) = Q_3(p)$ and $Q_2(p) = Q_4(p)$*

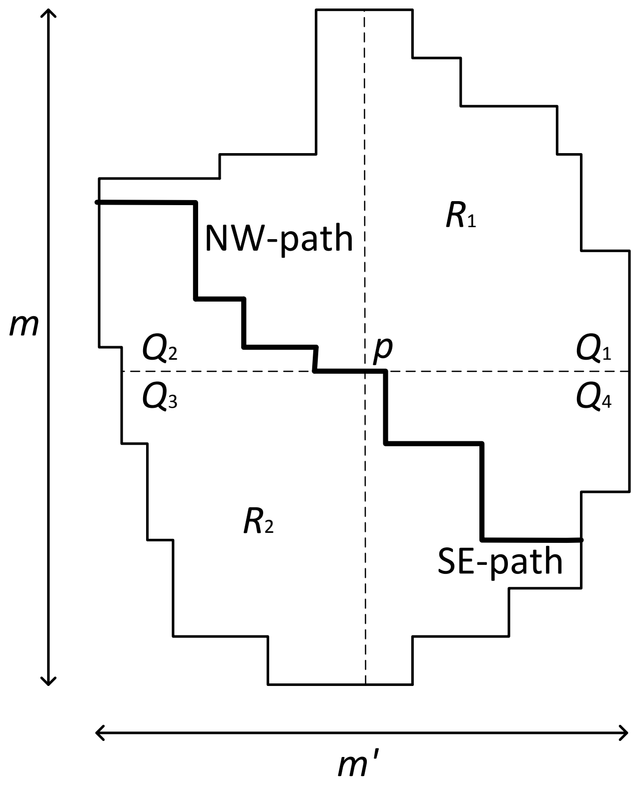


Figure 2.6: Partitioning an area for efficient exploration

2. $Q_1(p) + Q_3(p) \geq A(R)/2$ *or* $Q_2(p) + Q_4(p) \geq A(R)/2$.

Proof: Consider the function $f_{13}(x, y) = Q_1((x, y)) - Q_3((x, y))$. If x is smaller than any x-coordinate of a point in R and y is smaller than any y-coordinate of a point in R, then $f_{13}(p_x, p_y) = A(R)$. If x is larger than any x-coordinate of a point in R and y is larger than any y-coordinate of a point in R then $f_{13}(p_x, p_y) = -A(R)$. Further, the function is continuous and decreases with x and y. Therefore, for each x there exists a y such that $f_{13}(x, y) = 0$ and for all y there exists a x such that $f(x, y) = 0$.

For the function $f_{24}(x, y) = Q_2((x, y)) - Q_4((x, y))$, we can deduce the equivalent observations.

Given a rectangle $(x_1, y_1), (x_2, y_2)$ where $f_{13}(x_1, y_1) \leq 0$, $f_{13}(x_2, y_2) \geq 0$, $f_{24}(x_2, y_1) \leq 0$ and $f_{24}(x_1, y_2) \geq 0$, we can conclude that in one of the four equal-sized sub-rectangles this condition is preserved. The choice of the rectangle depends on the signs of $f_{13}(\frac{1}{2}(x_1 + x_2), \frac{1}{2}(y_1 + y_2))$ and $f_{24}(\frac{1}{2}(x_1 + x_2), \frac{1}{2}(y_1 + y_2))$. This implies the existence of a point p where $f_{13}(p_x, p_y) = f_{24}(p_x, p_y) = 0$.

Since $f_{13}(p_x, p_y) = f_{24}(p_x, p_y)$ it follows $Q_1(p) = Q_3(p)$ and $Q_2(p) = Q_4(p)$. If $Q_1(p) + Q_3(p) \geq Q_2(p) + Q_4(p)$ then

$$2(Q_1(p) + Q_3(p)) \geq Q_1(p) + Q_3(p) + Q_2(p) + Q_4(p) = A(R) \ .$$

Otherwise we have $Q_1(p) + Q_3(p) < Q_2(p) + Q_4(p)$ and therefore

$$2(Q_2(p) + Q_4(p)) > Q_1(p) + Q_3(p) + Q_2(p) + Q_4(p) = A(R) \ .$$

□

This point p can be efficiently computed, since the region R is defined by horizontal boundaries. Now we navigate a robot to this point p using the algorithm of [BEBFY94]. If it lies within an obstacle, the algorithm will circle the obstacle, otherwise it will reach the node which is in smallest distance to the point.

Assume that $Q_1(p) + Q_3(p) \geq A(R)/2$. Then, we will construct a path within the second and forth quadrant which divides R into R_1 and R_2. For this, starting from p we simply follow a greedy NW-path until we reach the boundary of R. Then, we follow a greedy SE-path starting from p until we reach the boundary of R. This path will not leave the second and forth quadrant and the sub-regions are again *surroundable*.

If p lies within an obstacle, we take the obstacle corner points in the second and forth quadrant with respect of p. Then, we construct NW-paths and SE-paths from these two points and combine them with two surrounding paths of the rectangular obstacles around p.

In both cases we have $A(R_1) \geq Q_1(p) \geq \frac{1}{4}A(R)$ and also $A(R_2) \geq \frac{1}{4}A(R)$.

If $Q_1(p) + Q_3(p) < A(R)/2$ then we have $Q_2(p) + Q_4(p) \geq A(R)/2$ and we make the symmetric construction within the first and third quadrant of p using greedy NE and SW-paths. Again, we get $A(R_1) \geq \frac{1}{4}A(R)$ and $A(R_2) \geq \frac{1}{4}A(R)$. Since $A(R_1)+A(R_2) = A(R)$ the claim follows. □

Algorithm 2 uses this partitioning to explore the square.

Algorithm 2: $\mathcal{O}(\log^2 m)$-competitive multi-robot exploration of the $m \times m'$ grid with k robots

```
1: Start with the full square as a single surroundable region
2: All robots start in the upper left corner
3: for $i \leftarrow 1, 2, \ldots, \log k$ do
4:     Partition all $2^{i-1}$ regions in parallel using one robot per region
5: end for
6: while Unexplored regions exist do
7:     Explore all $k$ regions with one robot each using depth-first search
8:     If a robot finishes the DFS it returns to the upper left corner
9:     if at least $k/2$ robots have returned to the upper left corner then
10:        Stop the entire exploration
11:        Partition all $k/2$ unexplored regions
12:    end if
13: end while
```

Theorem 4 *Algorithm 2 can explore the $m \times m'$ grid with k robots in time $\mathcal{O}(m \log^2 m + m \log m \log k + (n \log n)/k)$ where $m < n \leq m^2$ is the number of nodes in the grid (without obstacles) using the global communication model.*

Proof: First note that a single robot can explore a connected area with n' (non-obstacle) vertices using depth-first search in time $2n'$, but that such an area cannot be explored with less than n' steps by a single robot.

It takes at most $2 \log_{4/3} m$ rounds of re-partitioning until all surroundable regions have a size of at most 1, since the size of a region is reduced by at least a factor of 3/4. Each partitioning takes $\mathcal{O}(m \log m)$ steps for one robot. Moving to the left upper corner takes $2m - 1$ steps using a greedy NE-path. Hence, the time of the lines 3–5 can be estimated by $\mathcal{O}(m \log m \log k)$ steps. The while-loop (line 6) is executed at most $\mathcal{O}(\log m)$ times. All partitioning steps in line 11 take therefore $\mathcal{O}(m \log^2 m)$ steps.

For the exploration time, we consider the rounds of the while-loop (lines 6–13). Let n_j denote the number of unexplored vertices at the beginning of the j-th round. There are two cases:

In the first case, the loop finishes in round j since less than $k/2$ robots have returned and all regions are explored. So, more than $k/2$ robots have explored n_j vertices in parallel. This has taken at most $4n_j/k$ steps, since $k/2$ robots have explored at most n_j vertices in parallel with DFS. Therefore, the time for this round can be estimated by $4n/k$.

In the second case, the $k/2$ robots have returned in the j-th round, but $k/2$ unexplored regions will be again completely revisited in the round $j + 1$. Let R be an explored region with the largest number of vertices (given by F). This means R is explored in at least $F(R)$ steps. All explored regions have been explored in at most $2F(R)$ steps. However, all $k/2$ unexplored regions must have had at least $\frac{1}{2}F(R)$ vertices, since otherwise they would have been explored in this round by the DFS. Summarizing over the $k/2$ unexplored regions we have $\frac{1}{4}kF(R) \leq n$ and therefore $F(R) \leq 4n/k$ which results in an upper time bound for the exploration of $8n/k$ steps in this round.

Since there are at most $\mathcal{O}(\log m)$ rounds in the while-loop there are at most $\mathcal{O}((n \log m)/k)$ steps for the exploration. □

The global communication of our model can be replaced with a local communication scheme, if all robots stop the algorithm every $8m$ steps, move to the left upper node, communicate, and then return to their work. This needs $4m$ steps and only increases the exploration time by a constant factor.

Corollary 2 *Using only local communication the $m \times m'$ grid can be explored with k robots in time $\mathcal{O}(m \log^2 m + m \log m \log k + (n \log m)/k)$, the same as using the global communication model.*

Every optimal exploration strategy where all k robots start in the left upper corner needs at least $2m - 1$ steps to reach the opposite corner. The other lower bound of n/k results from the optimal parallelization of the exploration of the n cells. Further note, that $k = m \cdot m'$ robots can explore the $m \times m'$ grid in time $\mathcal{O}(m \log m')$. For this purpose, each robot navigates to its assigned node. This establishes a competitive factor of $\mathcal{O}(\log m')$. For $k \leq m^2$ we have $\log k \leq 2 \log m$ and thus a run-time of $\mathcal{O}(m \log^2 m + n(\log m)/k)$ compared to lower bound of $\Omega(m + n/k)$ resulting in the following corollary.

Corollary 3 *There is an exploration strategy to explore an $m \times m$-grid and $m < n \leq m^2$ with oriented disjoint rectangular obstacles with a competitive exploration time ratio of $\mathcal{O}(\log^2 n)$ in the local communication model.*

2.4 Efficient Tree Exploration

In this section, we turn our eyes again back to the tree exploration. First, we introduce the Yo-yo algorithm. The Yo-yo algorithm perfectly parallelizes the work in a tree with any number of robots, but is very susceptible to the depth of a tree. The Yo* later is then an iterated version of the Yo-yo that can handle trees of larger depth efficiently.

The Yo-yo Exploration

The basic idea of the Yo-yo exploration algorithm is to successively explore every set of nodes in the tree with the same depth. After each exploration step all robots return to the root and are perfectly rebalanced for the next exploration step. For most trees, this algorithm is not very efficient, since most of the time the robots commute between the root and the leafs of the so far known sub-tree. We denote the number of nodes in depth i by n_i.

Algorithm 3: The Yo-yo Algorithm: $4d$-competitive multi-robot exploration of a tree

```
 1: All robots start at the root of the tree
 2: for $i \leftarrow 2, \dots, d$ do
 3:   Partition all $n_i$ nodes in depth $i$ into $k$ subsets $V_{i,1}, \dots, V_{i,k}$ with $|V_{i,j}| \leq \lceil \frac{n_i}{k} \rceil$.
 4:   for all $j \leftarrow 1, \dots, k$ do in parallel
 5:     for all $u \in V_{i,j}$ do
 6:       Move robot $j$ to $u$
 7:       Move robot $j$ to the root
 8:     end for
 9:   end for
10: end for
```

The main motivation of this algorithm is that the competitive ratio $4d$ only depends on the depth d, which we can improve later on by a technique which does not work with a competitive factor depending on k. Note that for at least $d \in \mathcal{O}(\frac{k}{\log k})$ robots Yo-yo is asymptotically at least as good as the best known algorithm of Fraigniaud et al. [FGKP06].

Lemma 11 *The Yo-yo algorithm needs at most $d(d+1) + 2dn/k$ rounds to explore a graph with n nodes, depth d and k robots, and thus has a competitive exploration ratio of at most $4d$.*

Proof: The success of the exploration algorithm follows by an easy induction over the tree depth. For the number of rounds, note that in lines 6 and 7 each of the k robots moves for $2i$ rounds in order to explore a node in $V_{i,j}$ and return to the root. This is repeated in the loop starting at line 5 for at most $\lceil n_i/k \rceil$ times. Therefore, the overall number of rounds for the outer loop starting at line 2 is the following.

$$\begin{aligned}
\sum_{i=1}^{d} 2i \left\lceil \frac{n_i}{k} \right\rceil &\leq \sum_{i=1}^{d} 2i \left(1 + \frac{n_i}{k}\right) \\
&= d(d+1) + \sum_{i=1}^{d} 2i \frac{n_i}{k} \\
&\leq d(d+1) + 2d\frac{n}{k} \ ,
\end{aligned}$$

where we use $\sum_{i=0}^{d} n_i = n$. Now, every exploration algorithm needs at least

$\max\{d, \lceil n/k \rceil\} \geq \frac{1}{2}(d + n/k)$ rounds. So, the competitive factor is at most

$$\frac{d(d+1) + 2d\frac{n}{k}}{\max\{d, \lceil n/k \rceil\}} \leq \frac{2d^2 + 2d\frac{n}{k}}{\frac{1}{2}(d + n/k)} \leq 4d$$

□

By ignoring any newly discovered edges leading to circles, the Yo-yo can also be adapted to graphs which are not trees. Because of the breadth-first order of exploration of Yo-yo the exploration time will remain unchanged for the graph exploration.

The Yo* Algorithm

Starting from the Yo-yo algorithm (Algorithm 4) we use a recursive approach to improve the efficiency of the exploration. To avoid the rebalancing step passing the root in each step we divide the graph into the uppermost segment of depth c and b segments of depth a such that $d = ab + c$ which values are to be chosen later on, see Figure 2.7. The first segment will be explored by the base algorithm, e.g. the Yo-yo algorithm. One can easily see that if the competitive ratio grows with the depth of the tree we can bound the ratio with a smaller term now.

All deeper unexplored segments will be handled together with the last explored segment (see Figure 2.8). These two segments form a forest of trees. If the number of trees is greater than the number of robots, we can use DFS to efficiently explore them. However, the size of the trees can differ and therefore, we rebalance the robots if half of the trees have been explored by DFS. The rebalancing costs at most d steps and this has to be repeated at most $\log n$ times.

If the number of trees has been reduced to be smaller than the number of robots, we use the base algorithm and rebalance again, if half of the trees have been completely explored. So, we have $\log k$ iterations for all the b segments.

Taking the Yo-yo algorithm and choosing segments of depth $a = c = d^{\frac{1}{2}}$ a back-on-the-envelope calculation gives us a competitive ratio of $\mathcal{O}(d^{\frac{1}{2}})$ for the first segment and a ratio of $\mathcal{O}(d^{\frac{1}{2}}(\log n + \log k))$ for all the other segments. So, after one iteration of the Yo* algorithm we improve the depth-dependent factor in the ratio from d to $d^{\frac{1}{2}}$. Now, if we take this new algorithm as base algorithm and choose segments of size $d^{\frac{1}{3}}$ we improve the ratio to $d^{\frac{1}{3}}$. However, there is an overhead in the iteration, where constant factors grow exponentially over the number of iterations and thus must be carefully analyzed.

We define $g_{\mathcal{A}}(d, k)$ to describe the competitiveness of an algorithm $\mathcal{A}$ that only depends on d and k for any tree. Now, we assume that we start from a $g_{\mathcal{A}}(d, k)(d + \frac{n}{k})$ time bounded algorithm and try to turn it into a more efficient one using the Yo* algorithm. From $g_{\mathcal{A}}(d, k)$ we only know that it is a monotone increasing function with respect to d and k, e.g. for the Yo-yo algorithm we have $g_{\text{Yo-yo}}(d, k) = 4d$. We will omit $\mathcal{A}$ on g for brevity when ever it does not lead to ambiguity.

Algorithm 4: The Yo* algorithm using a base algorithm

```
1:  All $k$ robots start at the root of the tree
2:  Explore the subtree of depth $c$ with the base algorithm
3:  for $j \leftarrow 1, \dots, b$ do
4:    $R \leftarrow$ set of nodes in depth $\max\{0, c+(j-2)a\}$, which are ancestors
          to at least one unexplored succeeding node in depth $[c+(j-1)a, c+ja]$
5:    while $R \neq \emptyset$ do
6:      if $k \leq |R|$ then
7:        Equally partition all nodes in $R$ into sets $V_1, \dots, V_k$ such that $|V_i| \leq \left\lceil \frac{|R|}{k} \right\rceil$
8:        for $i \leftarrow 1, \dots, k$ do
9:          $T_i \leftarrow$ minimum tree connecting all unexplored nodes in
                depth $[c+(j-1)a, c+ja]$ with an ancestor in $V_i$
10:       end for
11:       while less than $k/2$ subtrees of $R$ are explored do
12:         for all $i \leftarrow 1, \dots, k$ do in parallel
13:           Perform a DFS exploration step in $T_i$ with robot $i$
14:         end for
15:       end while
16:     else
17:       for $i \leftarrow 1, \dots, |R|$ do
18:         Equally assign $k_i$ robots to node $v_i$ of $R$ such that $k_i \in \left\{ \left\lfloor \frac{k}{|R|} \right\rfloor, \left\lceil \frac{k}{|R|} \right\rceil \right\}$
19:         $T_i \leftarrow$ minimum tree connecting all unexplored nodes in
                depth $[c+(j-1)a, c+ja]$ with ancestor $v_i$
20:       end for
21:       while less than $k/2$ subtrees of $R$ are fully explored do
22:         for all $i \leftarrow 1, \dots, |R|$ do in parallel
23:           Perform one step of the base exploration algorithm
              on $T_i$ with $k_i$ robots
24:         end for
25:       end while
26:     end if
27:     $R \leftarrow$ set of nodes in depth $\max\{0, c+(j-2)a\}$, which are ancestors
            to at least one unexplored succeeding node in depth $[c+(j-1)a, c+ja]$
28:   end while
29: end for
```

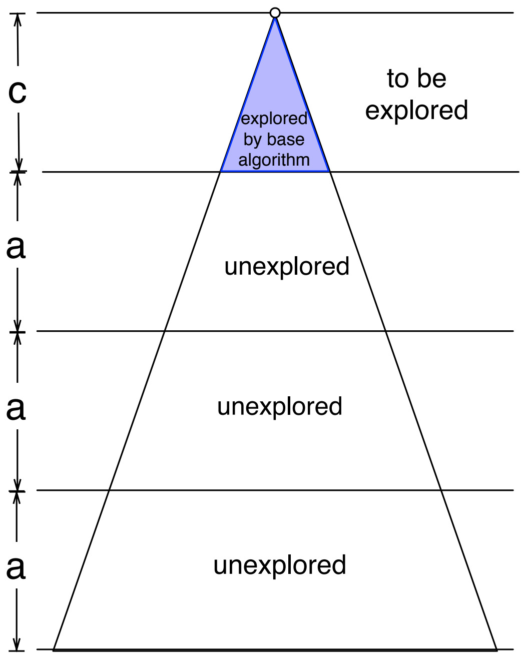


Figure 2.7: The first round of the Yo* algorithm

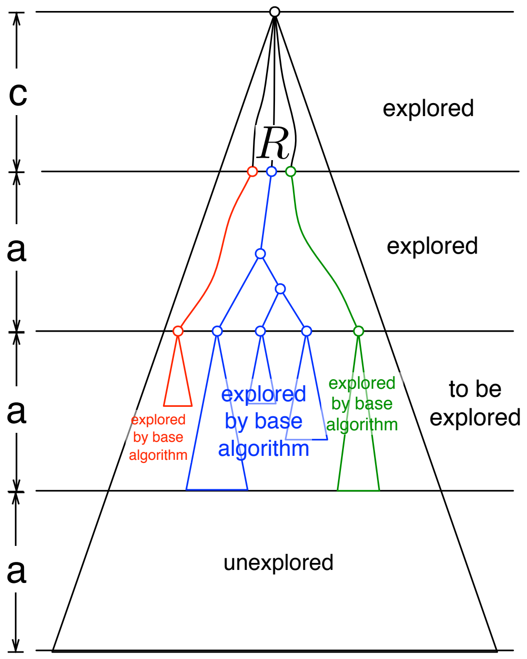

c
a
a
a
R
explored
explored
explored by base algorithm
explored by base algorithm
explored by base algorithm
to be explored
unexplored

Figure 2.8: The principle of the Yo* algorithm

Lemma 12 *Given a $g(d,k)(d+\frac{n}{k})$-time bounded base algorithm for a graph with unknown number of nodes n, given depth $d = ab + c$, $a, b, c \in \mathbb{N}$, and k exploring robots, then the Yo* algorithm can explore such a tree within the following number of rounds*

$$\left(d + \frac{n}{k}\right)(8g(2a,k)\log k + g(c,k) + 2b(\log k + \log n) + 4\log n) \enspace .$$

Proof: We denote by n_0 the number of nodes in depth at most c. By n_j we denote the number of nodes of the tree with depth in the interval $[1 + c + (j-1)a, c + ja]$. By definition $\sum_{j=0}^{b} n_j = |V|$.

We use the base algorithm to explore the first segment, which needs at most

$$t_1 = g(c,k)\left(d + \frac{n_0}{k}\right) \tag{2.1}$$

rounds.

In all other rounds we use the base algorithm several times when $k > |R|$. After each iteration of the while-loop from lines 21–25 the number of $|R|$ is reduced by a factor of 2, which implies at most $\log k$ iterations. The variable $\nu = 1, \ldots, \log k$ counts the iterations of this loop. Let $R_{j,\nu}$ be the variable R in the j-th loop and the ν-th iteration. Let $k_{j,\nu}$ be the smallest number of robots in this phase, i.e. $k_{j,\nu} = \lfloor k/|R_{j,\nu}| \rfloor$.

The trees connecting all unexplored nodes with an ancestor node in $R_{j,\nu}$ are named $T_{j,\nu,i}$ for $i \in \{1, \dots, |R_{j,\nu}|\}$. Now define

$$n_{j,\nu} := \text{median}(|V(T_{j,\nu,i})| \ , \ i \in \{1, \dots, |R_{j,\nu}|\})$$

where for an even number m the median refers to the $m/2$-th largest element. Note that the median implies that

$$n_{j,\nu} \frac{|R_{j,\nu}|}{2} \leq \sum_{i=1}^{|R_{j,\nu}|} |V(T_{j,\nu,i})| \leq n_{j-1} + n_j$$

So, we can conclude that

$$\sum_{\nu=1}^{\log k} n_{j,\nu} |R_{j,\nu}| \leq 2(n_j + n_{j-1}) \log k \ .$$

The run-time of one invocation the base algorithm is by definition at most

$$g(2a, k_{j,\nu}) \left(2a + \left\lceil \frac{n_{j,\nu}}{k_{j,\nu}} \right\rceil \right)$$

by design of the loop in line 21. Since $k_{j,\nu} \geq |R_{j,\nu}|$ and $n_{j,\nu} \geq 1$ we can use $\left\lceil \frac{x}{\lfloor y \rfloor} \right\rceil \leq 2\frac{x}{y} + 1$ for $x, y \geq 1$.

$$\left\lceil \frac{n_{j,\nu}}{k_{j,\nu}} \right\rceil = \left\lceil \frac{n_{j,\nu}}{\lfloor k/|R_{j,\nu}| \rfloor} \right\rceil \leq 2\frac{n_{j,\nu}|R_{j,\nu}|}{k} + 1$$

The run-time over all invocations of all these loops is therefore

$$\begin{aligned}
t_2 &\leq \sum_{j=1}^{b} \sum_{\nu=1}^{\log k} g(2a, k_{j,\nu})(2a + \lceil n_{j,\nu}/k_{j,\nu} \rceil) \\
&\leq g(2a, k) \left(2ab \log k + b \log k + 2 \sum_{j=1}^{b} \sum_{\nu=1}^{\log k} \frac{n_{j,\nu}|R_{j,\nu}|}{k} \right) \\
&\leq g(2a, k) \left(2ab \log k + b \log k + 2 \sum_{j=1}^{b} \frac{2(n_{j-1} + n_j) \log k}{k} \right) \\
&\leq g(2a, k) \left(2(d - c) \log k + b \log k + 8\frac{n}{k} \log k \right) \\
&\leq \left(2d + b + 8\frac{n}{k} \right) g(2a, k) \log k \ .
\end{aligned}$$

It remains to count all rebalancing moves of the robots. It takes at most $2d$ steps to reassign a robot to its new tree. For the case $k > |R|$, this iterates at most $b \log k$ times, resulting in

$$t_3 \leq 2bd \log k$$

steps.

Now we analyze the case $k \leq |R|$. After each iteration of the loop of line 11 the number of nodes in $|R|$ is halved. Hence, the number of loops is bounded by $\log n$. The sum of all iterations of the loop 11 is bounded by $4(n_{j-1} + n_j)/k$ rounds, since $k/2$ robots successfully explore the graph in parallel. So, summing over all j we get

$$t_4 \leq \sum_{j=1}^{b} \frac{4(n_{j-1} + n_j)}{k} \leq 8\frac{n}{k}\log n$$

for the DFS-exploration. Again we have to rearrange the robots between the trees which costs at most

$$t_5 \leq 2bd \log n$$

additional steps.

So, for $c \leq 2a$ and $d \geq 1$ the time-cost is bounded by:

$$\begin{aligned} t_f &= t_1 + t_2 + t_3 + t_4 + t_5 \\ &\leq g(c,k)\left(d + \frac{n_0}{k}\right) + g(2a,k)\left(2d + b + 8\frac{n}{k}\right)\log k \\ &\quad + 2bd\log k + 4\frac{n}{k}\log n + 2bd\log n \\ &\leq d\left(g(c,k) + \left(2 + \frac{b}{d}\right) g(2a,k)\log k\right) \\ &\quad + \frac{n}{k}\left(g(c,k) + 8g(2a,k)\log k + 4\log n\right) + 2db(\log k + \log n) \\ &\leq \left(d + \frac{n}{k}\right)\left(g(c,k) + 8g(2a,k)\log k + 2b(\log k + \log n) + 4\log n\right) \end{aligned}$$

□

We use polynomials of d for a and b, which results in the following Lemma.

Lemma 13 *Given a $g(d,k)(d + \frac{n}{k})$-time bounded base algorithm $\mathcal{A}$-ℓ for a graph with unknown number of nodes n, given depth d and k exploring robots, the Yo* algorithm provides a $(d+\frac{n}{k})(9g(2d^\alpha,k)\log k + 8d^{1-\alpha}(\log k + \log n))$-time bounded robot exploration algorithm $\mathcal{A}$-l+1.*

Proof: We choose $a = \lfloor d^\alpha \rfloor$, $b = \lfloor d/a \rfloor$, and $c = n - ab$. Note that $c \leq a$ and $b \leq 2d^{1-\alpha}$. From Lemma 12 it follows that the exploration time of Yo* is the following:

$$\begin{aligned} t_f &\leq \left(d + \frac{n}{k}\right)\left(g(c,k) + 8g(2a,k)\log k + 2b(\log k + \log n) + 4\log n\right) \\ &\leq \left(d + \frac{n}{k}\right)\Big(g(d^\alpha,k) + 8g(2d^\alpha,k)\log k + 4d^{1-\alpha}(\log k + \log n) + 4\log n\Big) \\ &\leq \left(d + \frac{n}{k}\right)\Big(9g(2d^\alpha,k)\log k + 8d^{1-\alpha}(\log k + \log n)\Big) \end{aligned}$$

□

Starting from the $4d$-competitive Yo-yo algorithm, or synonymously Yo-0 for the iteration, we choose $\alpha = \frac{1}{2}$ and obtain by the last Lemma a $\mathcal{O}(d^{\frac{1}{2}}(\log k + \log n))$-competitive multi-robot exploration algorithm. This algorithm can also be asymptotically improved by the same lemma. For this purpose we choose $\alpha = \frac{2}{3}$ and get a $\mathcal{O}(d^{\frac{1}{3}}(\log k)(\log k + \log n))$ algorithm. Of course this process can be iterated using the following lemma.

Lemma 14 *For $k \geq 2$, $c \geq 4$, $\beta \in [0, 1]$, $\gamma \geq 0$ and a base exploration algorithm Yo-ℓ with a run-time of $\left(d + \frac{n}{k}\right) cd^{\beta}(\log k)^{\gamma}(\log k + \log n)$, the Yo* algorithms can achieve an exploration time bound of $\left(d + \frac{n}{k}\right) 20c \cdot d^{\beta/(\beta+1)}(\log k)^{\gamma+1}(\log k + \log n)$.*

Proof: We choose $\alpha = \frac{1}{1+\beta}$ such that $\alpha\beta = 1 - \alpha$. This observation will be used for the run-time of an iteration the Yo* algorithm.

$$\begin{aligned}
g_{\text{Yo-}\ell+1} &\leq 9g_{\text{Yo-}\ell}(2d^{\alpha}, k)\log k + 8d^{1-\alpha}(\log k + \log n) \\
&\leq 9c2^{\beta}d^{\alpha\beta}(\log k)^{\gamma+1}(\log k + \log n) + 8d^{1-\alpha}(\log k + \log n) \\
&\leq 18cd^{\beta/(\beta+1)}(\log k)^{\gamma+1}(\log k + \log n) + 8d^{\beta/(\beta+1)}(\log k + \log n) \\
&\leq 20cd^{\beta/(\beta+1)}(\log k)^{\gamma+1}(\log k + \log n)\ ,
\end{aligned}$$

where we use $2^{\beta} \leq 2$ and $18c + 8 \leq 20c$ for $c \geq 4$.

□

Note that the iteration $\beta \mapsto \beta/(\beta + 1)$ with starting point $\beta = 1$ results in the series $1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \ldots$. Let $\beta_1 := 1$ and $\beta_{i+1} := \beta_i/(\beta_i + 1)$. If $\beta_i = \frac{1}{i}$, then

$$\beta_{i+1} = \frac{\frac{1}{i}}{\frac{1}{i} + 1} = \frac{1}{i+1}\ .$$

So, after ℓ iterations of the Yo* algorithm, starting from the Yo-yo algorithm we have the following ratio:

$$g_{\text{Yo-}\ell} \quad \leq \quad 4 \cdot 20^{\ell} d^{\frac{1}{\ell+1}}(\log k)^{\ell}(\log k + \log n)\ .$$

So far, we have assumed to know the depth. This is not necessary, since we use exponential doubling to find it. This introduces an additional factor of $\log d$ which we will take into account from now on. This competitive factor is only taken into account once, since after a correct guess the recursive approach invokes the next exploration algorithm with the correct depth value.

Theorem 5 *The Yo* multi-robot exploration algorithm with ℓ iterations can explore an unknown tree with depth d and size n with a competitive ratio of at most*

$$\mathcal{O}\left(20^{\ell} d^{\frac{1}{\ell+1}}(\log k)^{\ell}(\log k + \log n)(\log d)\right)\ .$$

Proof: Since the depth of tree is unknown we iteratively restart the Yo* exploration algorithm with an assumed depth of $d' = 1, 2, 4, \ldots$. An exploration is canceled if a node with depth larger than d' has been found, then the exploration starts from scratch. In the final step the time for the exploration is therefore at most (assuming $d' = 2d - 1$ in the worst case)

$$\left(2d - 1 + \frac{n}{k}\right) 4 \cdot 20^{\ell}(2d-1)^{\frac{1}{\ell+1}}(\log k)^{\ell}(\log k + \log n) \ .$$

Now $2d - 1 + n/k \leq 2(d + n/k)$ and $(2d-1)^{\frac{1}{\ell+1}} \leq (2d)^{\frac{1}{\ell+1}} \leq 2d^{\frac{1}{\ell+1}}$ results in an additional factor of 4. It takes $\log d$ iterations until $d' \geq d$ and therefore we have a total run-time of at most

$$16\left(d + \frac{n}{k}\right) \cdot 20^{\ell} d^{\frac{1}{\ell+1}}(\log k)^{\ell}(\log k + \log n)\log d$$

The competitive factor originates from the observation that the minimal time for offline exploration is $\max\{d, n/k\} \geq \frac{1}{2}(d + n/k)$.

$\square$

This is our main important result for tree exploration algorithms. What follows is a discussion of how many iterations are necessary to achieve best possible asymptotic bounds. It turns out that the relationship between the depth and the number of robots is crucial. For very small depths $d = \mathcal{O}((\log n)^c)$, already the Yo-yo algorithm provides a competitive ratio of $\mathcal{O}((\log n)^c)$. Similar for small poly-logarithmic teams of robots $k = \mathcal{O}((\log n)^c)$ a single robot doing DFS will achieve a bound of $\mathcal{O}((\log n)^c)$. For polynomial depths and robots, as we require for our model, Yo* provides better bounds:

Theorem 6 *The Yo* algorithm can achieve a competitive factor of*

$$2^{(2+o(1))\sqrt{(\log d)(\log\log k)}}(\log k)(\log k + \log n)$$

for a k-multi-robot exploration of graphs of size n and depth d.

Proof: Again we test the depth of the tree by performing the ℓ iterations of the Yo* algorithm, where we double a depth parameter d' every time we finde a node in depth $d' + 1$. Then, we relaunch the exploration. As the iteration depth of Yo* we choose $\ell = \left\lceil \sqrt{\frac{\log d'}{\log\log k}} \right\rceil$, because

$$(\log k)^{\ell} = 2^{\left\lceil \sqrt{\frac{\log d'}{\log\log k}} \right\rceil \log\log k} \leq 2^{\sqrt{(\log d')(\log\log k)}} \log k$$

and

$$d'^{2/(2\ell+1)} \leq 2^{(\log d)\sqrt{\log\log k}/\sqrt{\log d}} = 2^{\sqrt{(\log d)(\log\log k)}}$$

Now we have

$$20^{\ell}(\log d') \leq 2^{\sqrt{\log d}+\log\log d} = 2^{o(1)\sqrt{(\log d)(\log\log k)}}$$

for large enough k. So, the only remaining relevant factor is $\log k + \log n$ which implies the result.

$\square$

This bound is not always smaller than the best known competitive ratio of $\mathcal{O}(k/\log k)$. Yet, for trees with depth $d \in \mathcal{O}(n/k)$ and $k = n^c$ for some $0 < c < 1$ this becomes a bound of:

$$\mathcal{O}(2^{\sqrt{(\log n)(\log\log n)}}(\log^2 n)) = n^{o(1)}$$

2.5 Empirical Evaluation of Tree Algorithms

We present here a short empirical evaluation of the Yo-yo and Yo* algorithm. We use a computer simulation to explore a comb-structured tree. The comb structure is chosen as it works as a lower bound for greedy algorithms and at the same time punishes algorithms that are not greedy enough. For example, an algorithm only sending a new robot from S when ever a node has more than 2 children will be $\mathcal{O}(d)$-competitive.

For the exploration, we fix the size of the tree to the number of robots. Accordingly, if we explore a tree with k robots the tree will have $n = k^2$ nodes and a depth of $d = 2k - 2$ (see Figure 2.9).

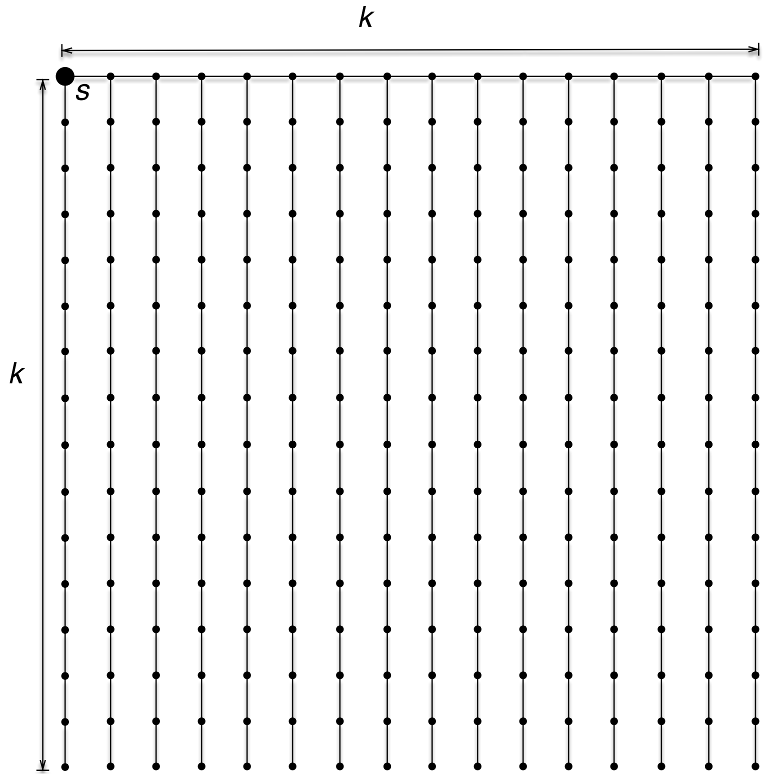


Figure 2.9: Comb scaled to number of robots

In Figure 2.10 we can see a comparison of the Yo-yo as well as Yo* algorithms with one, two and three iterations. The curve labeled as *Equal-dist* is the algorithm shown by Fraigniaud et al in [FGKP06]. It distributes robots equally to all child nodes in the tree and finishes all sub-trees before returning to a parent node.

The implementations of Yo-yo (Algorithm 3) and the Yo* (Algorithm 4) was done slightly different from the version used in the proofs. First of all, we fix the number of iterations of Yo* and name the resulting algorithm accordingly Yo-ℓ. The Yo-yo algorithm is in this nomenclature synonymous to Yo-0. We never execute a DFS step (lines 7–15 in Algorithm 4) in the Yo-ℓ algorithms. Since we are not using DFS as base algorithm, $k \geq |R|$ is assumed to be always true. Additionally instead of choosing R to depth $\max\{0, c + (j-2)a\}$, we set these root vertices of subtrees to depth $c + (j-1)a$ instead (line 4). Moreover we provide the algorithms with the depth of the tree beforehand so it does not have to restart.

From Figure 2.10 we can see that Yo-Yo and Yo-1 for all tested values of k need more steps than the other algorithms. While Yo-2, Yo-3 are on a par for the tested tree size.

In Figure 2.11 and Figure 2.12 the competitive values are plotted for better comparison. Viewing the exploration it becomes clear that constants dominate explorations of graph sizes which we were able to simulate. Further iterations of Yo* can not improve its speed, but will slow it down. The large competitive factors shown in Figures 2.11 and 2.12 indicate that the algorithm as it is, is not useful for real world application. For $k > 1000$ we can see the competitive factor is roughly by a factor of 2 better than an algorithm using only DFS with a single robot!

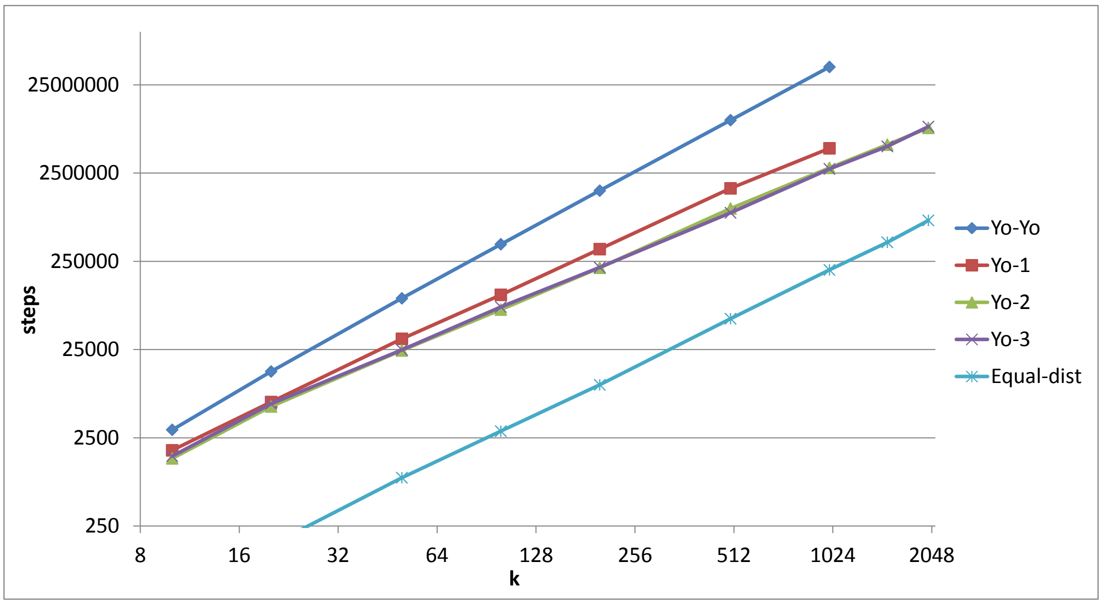


Figure 2.10: Exploration time for Comb

Several optimizations come to mind for improving exploration in practice.

1. Not stopping the exploration while redistributing robots could reduce the number of re-distributions in case of the comb from $\log k$ to 1 as all robots except the ones furthest to the right only have a line to explore.
2. Not waiting for half the trees being finished before redistributing robots, but instantly redistributing robots of finished trees could slightly improve the speed

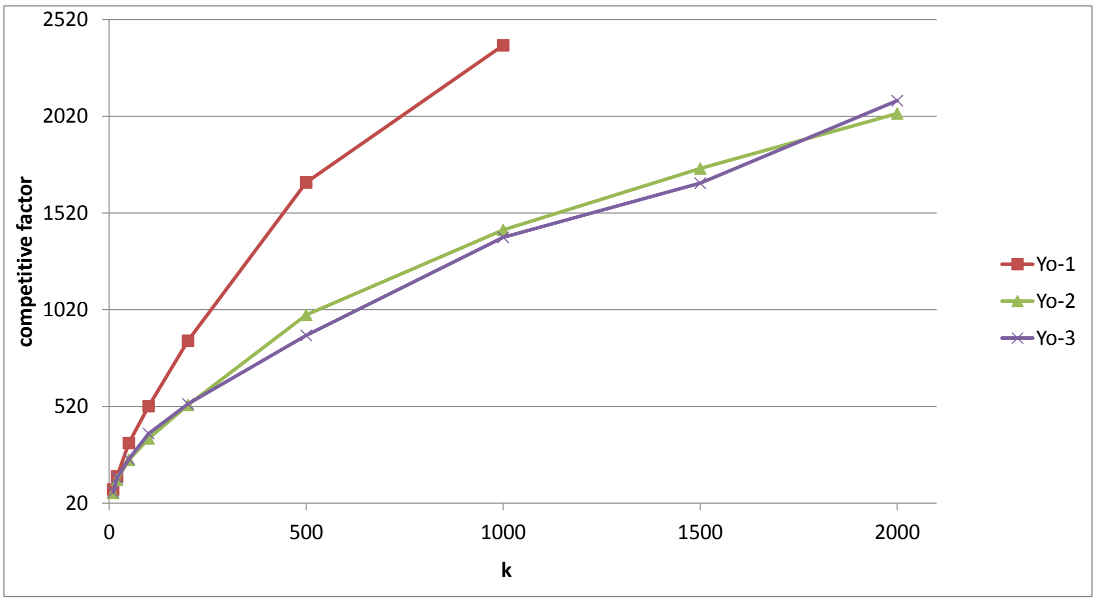


Figure 2.11: Competitive values for Comb

of the exploration if combined with optimization 1. .

3. A faster base algorithm than the Yo-yo could improve exploration. Not going back to the root, if enough robots reside in nodes with unexplored children, will change the base algorithm for exploring teeth of the comb from an $\mathcal{O}(d)$-competitive to an $\mathcal{O}(1)$-competitive algorithm.
4. Waiting for a level in the Yo* tree to finish before going to the next is necessary to concentrate robots in dense areas. However, going on with a single Robot executing DFS could improve exploration for trees like the comb slightly.

We are so far unable to proof if any or all of these improvements combined have an influence on the asymptotic run-time of the Yo* algorithm for general trees.

Yet, for further experiments we define a new algorithm called *Hedge Cutter*, by changing the Yo* to not stop while redistributing robots (optimization 1) combined with instantly redistributing the robots of finished trees. With this change the penalty for more iterations of this new *Hedge Cutter* algorithm is reduced, or as we conjecture, removed. Tests show that choosing $a = c = d/2, b = 1$ leads for the comb to the fastest run-time. If we iterate the algorithm indefinitely, it has also the benefit that knowledge of d beforehand becomes unnecessary. By choosing a and c in such a way that the algorithm separates the depth in half, it inherently does an exponential search for the depth while executing. Also, by iterating until a tree only consists of a single node, the optimization 3 has no further effect on the exploration. This means for example any sub-tree T, that is a path graph, is explored in $|T| = d = n_T$ steps.

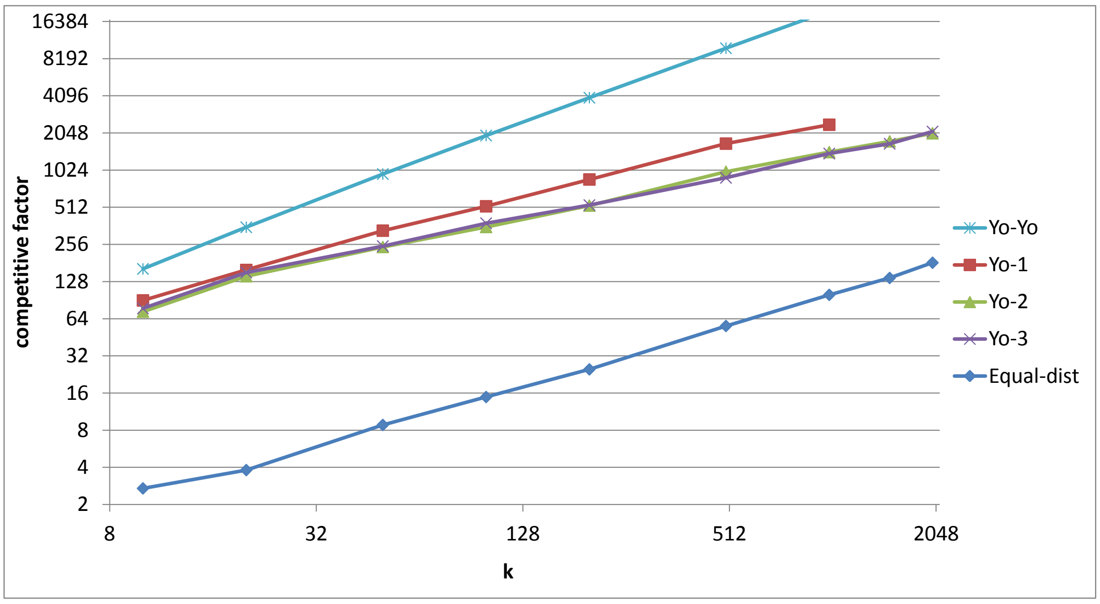


Figure 2.12: Competitive values for Comb in log-log

In Figure 2.13 we can see that this improvement greatly increases the speed. For comparison *Offline* shows the optimal time the comb can be explored in by an offline algorithm (2k-2 steps), while HC-ℓ shows different Hedge Cutter variants restricted to ℓ iterations. In the log-log plot (Figure 2.15) the downward slope of the curve for the Hedge Cutter algorithm is visible. That is exactly what we expect to see for an algorithm with a sub-polynomial run-time. The competitive factor on the largest comb we are able to simulate is smaller than 6. We can even proof that the Hedge Cutter has logarithmic competitiveness on the comb:

Theorem 7 *The Hedge Cutter algorithm can achieve a competitive factor of*

$$\mathcal{O}(\log d)$$

for a k-multi-robot exploration of a comb of size $n = m \cdot m$ and depth $d = 2m - 2$.

Proof:

Time for exploring the comb spine of length m is bounded by the time for the left comb spine half $T(m/2)$, the maximum time to move to a tooth (m), the time for the last or leftmost robot to explore a tooth and to move to the lower comb spine half $(2m)$ and to explore the lower comb spine half then with all robots $T(m/2)$ (see Figure 2.16):

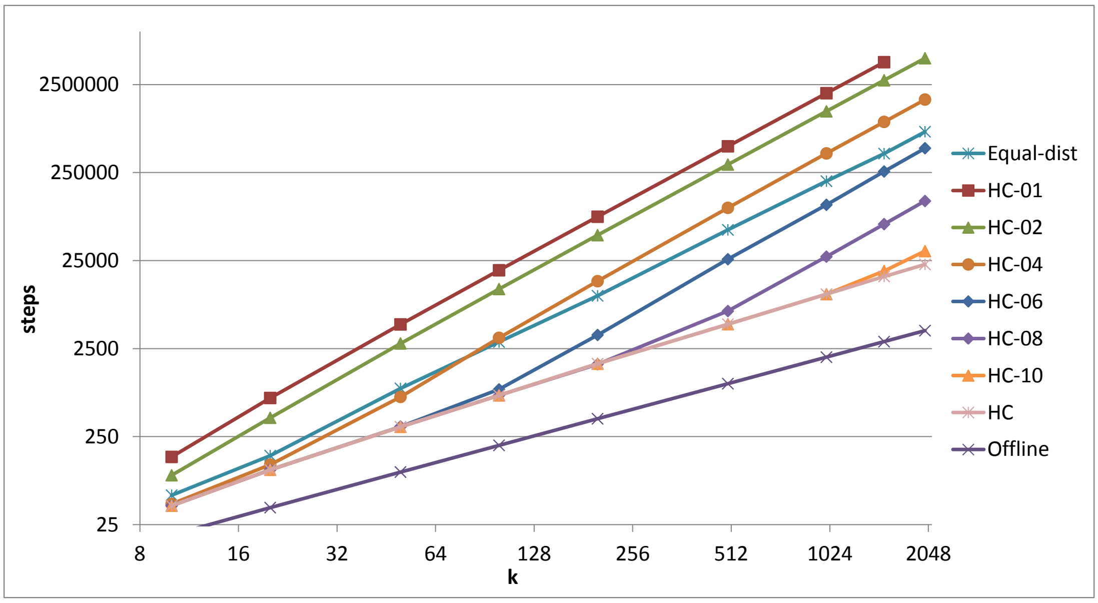


Figure 2.13: Exploration time for Comb with Hedge Cutter

$$\begin{aligned} T(m) &\leq T(m/2) + m + 2m + T(m/2) \\ &= 3\log(m)\cdot m \\ &= \mathcal{O}(m\log m) \end{aligned}$$

After the whole spine is explored, the rest of the comb will be explored in time $2m$. Therefore the Hedge Cutter is $\mathcal{O}(\log d)$ competitive on the comb.

□

We can not show a competitive bound for general trees, but conjecture that the exploration of a comb represents an asymptotic worst case for the Hedge Cutter.

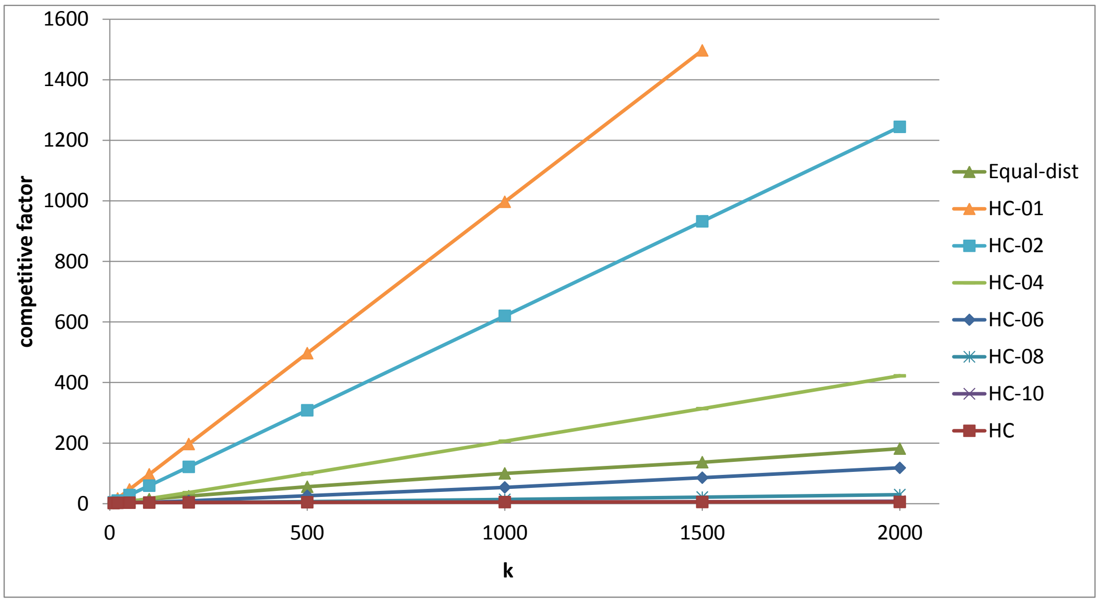


Figure 2.14: Competitive values for Comb with Hedge Cutter

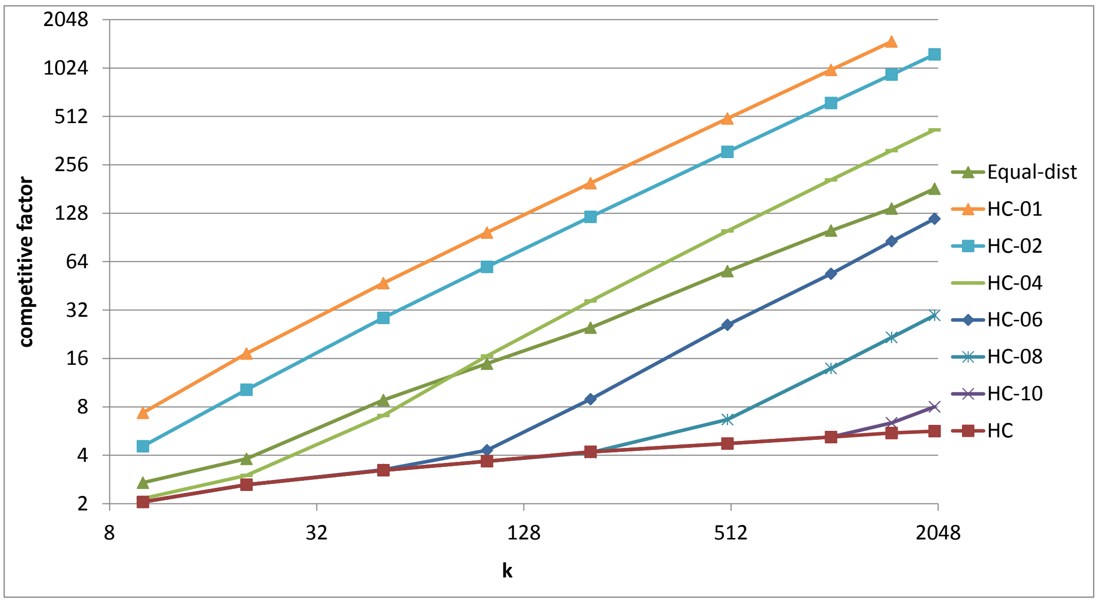


Figure 2.15: Competitive values for Comb in log-log

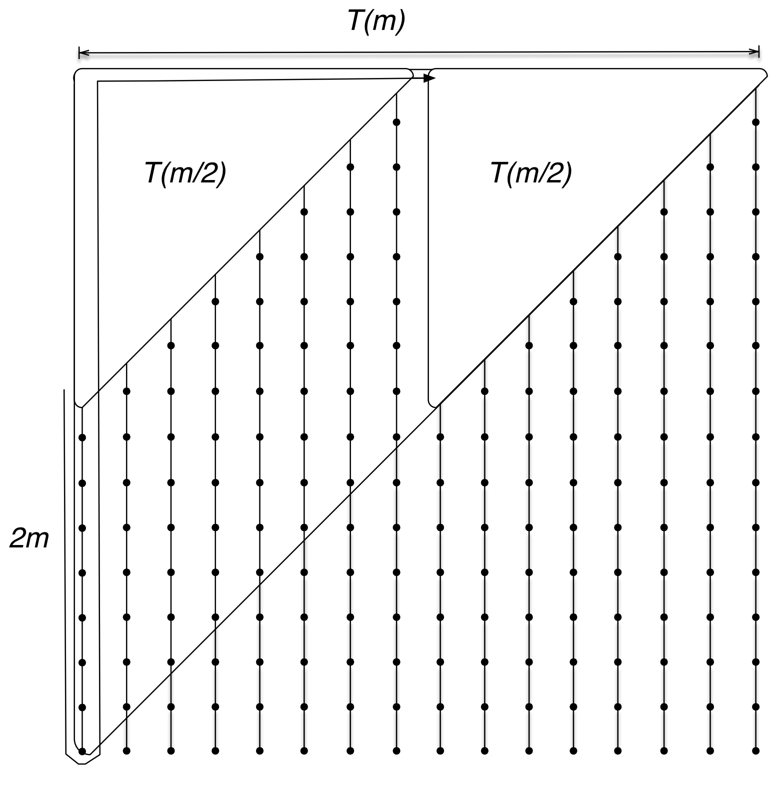


Figure 2.16: Behavior of Hedge Cutter on the comb

3 Parallel Unaware Cleaning

In this chapter, we will take a look at parallel unaware cleaning. Recall that this means that we provide the robots with complete knowledge of the graph, but instead forbid the online algorithm any knowledge about other robots set onto the same task. This includes any form of communication. Already knowing the starting positions of other robots in the graph would completely solve parallel unaware cleaning. Even measuring indirectly other robots, e.g. bumping into them or measuring the height of grass or amount of dirt on the path, could be used for communication and can therefore not be allowed in this setting.

As an illustration and starting example we show how differently a circle graph and a path graph behave in this setting(see Figure 3.1 and Figure 3.2). The simple algorithm sending robots in one direction in the circle (go right), or just to one end on the line, then returning to the other end (right-left traversal), performs quite differently for both graphs.

On the circle the go right traversal strategy performs very well. With robots distributed among the circle, the largest distance between these robots r will be the first visit time as well as the revisit time for this online strategy. It also provides a natural lower bound for the offline strategy. The first visit time t_f of the offline strategy can never be smaller than $t_f = r/2$. If the offline strategy uses both adjacent robots to clean this gap, while at the same time not leaving any other gap unattended, the right traversal strategy is at most by factor 2 slower than the offline algorithm.

For the path graph, the overhead of such an algorithm is a factor of n. If the left end node is not covered and all robots walk first to the right end and then return, no robot can visit the left node in less than n rounds. A smarter oblivious algorithm could improve this by sending robots into a random direction instead, yielding a competitive factor of $\mathcal{O}(\log n)$ in the expectation. However, a deterministic solution exists: the smart cow algorithm [BCR93], which in the i-th phase for $i = 1, 2, \dots, n$ explores 2^i nodes first to the left and then 2^i nodes to right from the starting node. While the smart cow algorithm is designed to find a hole in a fence, which it does within a competitive factor of nine, the same competitive factor can be shown for the cycle and the path graph. This shows that for these simple graphs deterministic competitive visiting strategies exist.

However, for the long term visit problem the situation is different. Symmetry cannot be resolved by any deterministic algorithm. If all robots have the same starting node no competitive ratio better than $\mathcal{O}(n)$ can be achieved for these algorithms.

The following section provides some simple algorithms and illustrates why these are

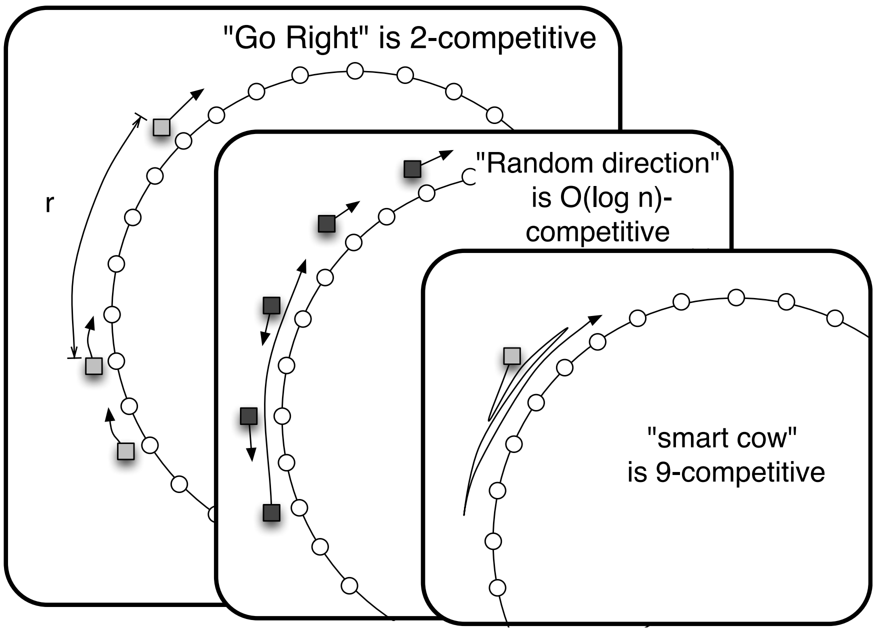

"Go Right" is 2-competitive
r
"Random direction" is O(log n)-competitive
"smart cow" is 9-competitive

Figure 3.1: Parallel unaware cleaning algorithms for the cycle graph. Illustrating competitive ratio for first visit.

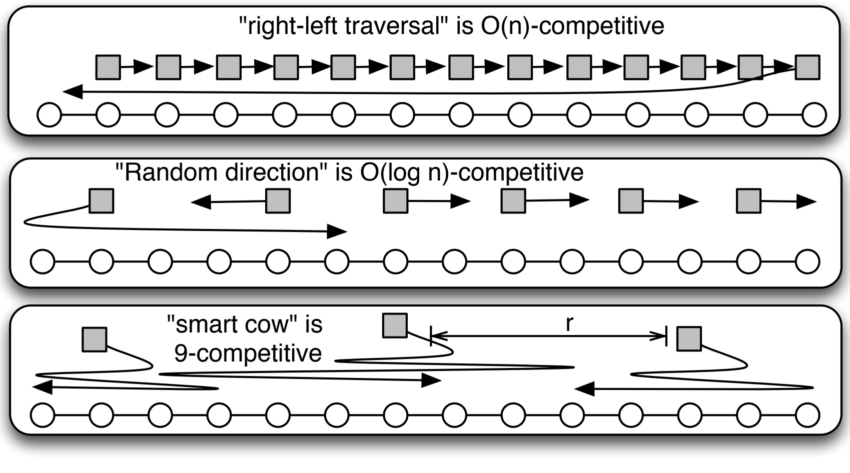

"right-left traversal" is O(n)-competitive
"Random direction" is O(log n)-competitive
"smart cow" is 9-competitive
r

Figure 3.2: Parallel unaware cleaning algorithms for the Path graph

not a solution to the visit problem.

3.1 Canonical Cleaning and General Observations

In the beginning of this section, we take a look at the Conscientious Reactive Algorithm (Algorithm 5). Machado et al. test in [MRZD03] several different architectures of communication between agents and perception of the environment. They show that the Conscientious Reactive Algorithm performs best in two empirical setting, consisting of two graphs of 50 nodes, 106 or 69 edges generated by putting obstacles in a grid graph and *skeletonizing* the remaining passable area. Between 1 to 30 robots with random starting positions were simulated over a large number of rounds.

With this algorithm, each robot has to keep track of the visit times of all nodes, but does not take the visits of other robots into account. The robot simply looks at its neighboring nodes and visits the one not visited for the longest time. By only choosing a random node for unvisited neighbors, we get a finite number of random choices. Since no communication with other robots takes place, this algorithm fits our own model and we can take it into account in Section 3.4 for experimental evaluation.

Algorithm 5: Conscientious Reactive Algorithm for Robot r

```
$v_r \leftarrow s_r$ starting position
while true do
    if exists by $r$ not visited node in $N(v_r)$ then
        $v_r \leftarrow$ choose not visited node by $r$ uniform at random from $N(v_r)$
    else
        $v_r \leftarrow$ node from $N(v_r)$ not visited for the longest time by $r$
    end
    Move to $v_r$
end
```

On a line or circle it behaves similarly to choosing a random direction at the beginning, i.e. while having a competitive first visit ratio of $\mathcal{O}(\log n)$, it is no solution to the long term visit problem as gaps will not be closed. But this $\mathcal{O}(\log n)$ ratio can not be proven for the first visit of every graph. To illustrate the need for more complex cleaning algorithms, we prove that the Conscientious Reactive Algorithm is uncompetitive.

Theorem 8 *Algorithm 5 is a high probability $\Theta(n)$-competitive first visit algorithm for undirected graphs.*

Proof: Consider the graph in Figure 3.3 consisting of a clique of size $\frac{n}{2}$ and $\frac{n}{2}$ single nodes, each connected only with one different node of the clique. Each robot starting at a node of the clique has for the rounds $t \leq \frac{n}{4}$ a probability of $p < \frac{4}{n}$ to visit a non-clique

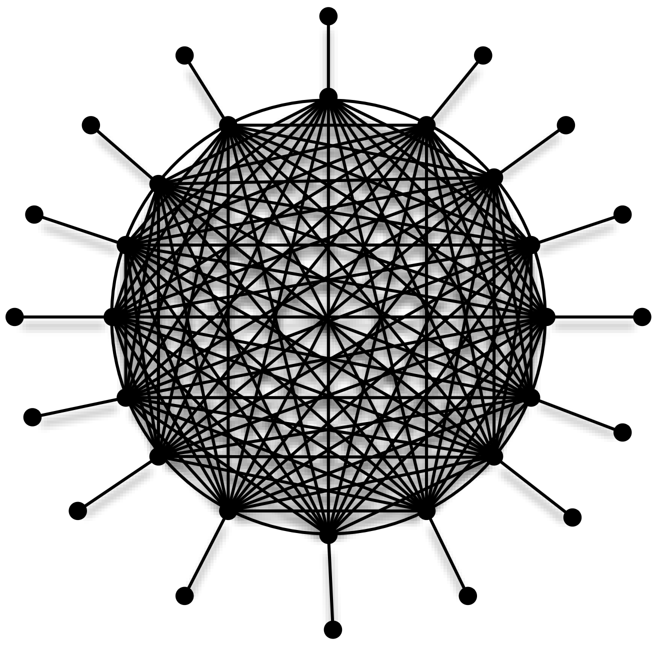


Figure 3.3: Worst case for Algorithm 5

node. So k robots will visit in expectation by round $n/4$:

$$E(\#visitednon - cliquenodes) < k \cdot \frac{n}{4} \cdot \frac{4}{n}$$

of the non clique nodes. So for $k \leq \frac{n}{2}$ robots in expectation not all non-clique nodes will have been visited and therefore there is no high probability for a first visit time of $t \leq \frac{n}{4}$. Therefore, Algorithm 5 is $\Omega(n)$-competitive.

Each robot will perform as worst case a DFS traversal of a graph and Algorithm 5 is therefore $\mathcal{O}(n)$-competitive. □

In Section 3.4, we will compare the Contentious Reactive Algorithm to solutions provided by us in the following pages.

Canonical Cleaning

Now, we present first general strategies and techniques that can be proven to work efficiently. For $u \in V$ let $N_\ell(u)$ denote the set of nodes in G within distance of at most ℓ to the node u. For a set $A \subseteq V$ let $N_\ell(A) = \bigcup_{u \in A} N_\ell(u)$. The following lemma is the key technique, which provides a lower bound for the number of robots in the vicinity.

Lemma 15 *Given a graph with a robot placement with a offline first visit time of t_f. Then, for any set of nodes A the number of robots in the node set $N_\ell(A)$ is at least*

$\lceil |A|/(t_f+1) \rceil$ for $\ell \geq t_f$.

Proof: First note that for each cleaning strategy it is not possible that robots outside of $N_{t_f}(A) \subseteq N_\ell(A)$ can reach any node within A in at most t_f steps. Let k be the number of robots that explore A within time t_f. At the beginning at most k nodes can be occupied by k robots. Then, in every subsequent round at most k additional nodes of A can be visited. In order to visit all nodes in A we have $k(t_f+1) \geq |A|$. This implies $k \geq \frac{|A|}{t_f+1}$. □

Later on, we use this lemma in a bait-and-switch strategy. We use A as bait to ensure that enough robots exist in a region for the offline strategy. Then we switch and let these robots work on other areas.

While randomization is necessary for dispersing the robots, too many probabilistic decisions are problematic, because the chance that some nodes remain unvisited for long times may grow over time. Therefore, we present only algorithms that use a finite number of randomized decisions. This technique is presented in the canonical algorithm, which is the base for some of our strategies. It requires the algorithms *cycle-start-node* and *waiting time* to provide where and when the robot should start cycling the graph.

Algorithm 6: Canonical cleaning algorithm for robot j using algorithms cycle-start-node and waiting-time

```
Traverse the graph by DFS yielding a cycle $P$ with $V(P) = V$ of length $2n$
$v_s \leftarrow$ cycle-start-node($s_j$)
Move robot $j$ on the shortest path to $v_s$
$w \leftarrow$ waiting time($s_j, v_s$)
Wait $w$ rounds
if $v_s$ occurs more than once in $P$ then
    Choose a random occurrence in P
end
while true do
    Walk to the next node of $P$
end
```

Because of the coupon collector's problem, a basic problem of probability theory [New60], one cannot expect a better competitive factor than $\mathcal{O}(\log n)$. Therefore, in the long run the problem can be solved by the canonical algorithm.

Theorem 9 *Using the* canonical cleaning *it is possible to achieve a long-term visit time of $\mathcal{O}((n/k)\log n)$ and a visit time of $\mathit{diameter}(G) + \mathcal{O}((n/k)\log n)$ with high probability.*

Proof: We choose for each robot an independent uniform random choice of the nodes of the cycle P as the *cycle-start-node*. The *waiting-time* is defined as $\mathit{diameter}(G) - |s_j, v_s|$. So, all nodes start the traversal at the same time.

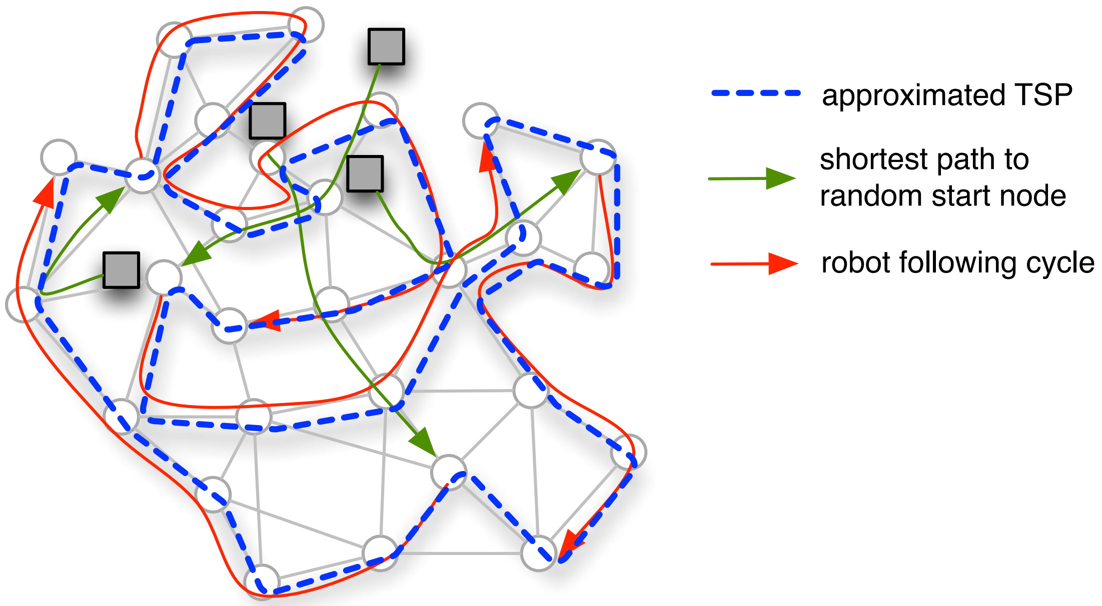


Figure 3.4: A canonical algorithm guarantees a $\mathcal{O}(\frac{n}{k}\log n)$ long-term visit time.

Let g be a subpath on the cycle P of length at most $2n$. The probability that no robots are in this subpath is $(1-\frac{g}{|P|})^k$. For k robots a subpath $g \geq \frac{2cn\ln n}{k}$ is empty with probability

$$\left(1-\frac{g}{|P|}\right)^k \leq \exp\left(-\frac{gk}{|P|}\right) \leq \exp\left(-\frac{gk}{2n}\right) \leq \exp\left(-c\ln n\right) \leq n^{-c} \,.$$

Hence, the maximum gap between two nodes on the cycle P is at most $\mathcal{O}((n/k)\log n)$ with high probability.

So, the long term visit time is bounded by this gap. From the waiting time, the first visit time follows. Note that after the first visit, the revisit time matches the long term visit time. $\square$

For graphs with small diameter this results in a logarithmic competitive ratio, e.g. in balanced trees the diameter is bounded by $\mathcal{O}(\log n)$. So, the CANONICAL CLEANING algorithm gives us the following bound.

Corollary 4 *Graphs with diameter of $\mathcal{O}(\log n)$ have a competitive ratio of $\mathcal{O}(\log n)$ for the first and revisit visit time with high probability.*

Proof: Let *cycle-start-node*(u) map to a uniform random node v of the tree. And let *waiting-time*(u,v) = *diameter*$(G) - |u,v|$. Let t_f^* and t_v^* be the optimal first and visit times and let $k \leq n$ be the number of robots.

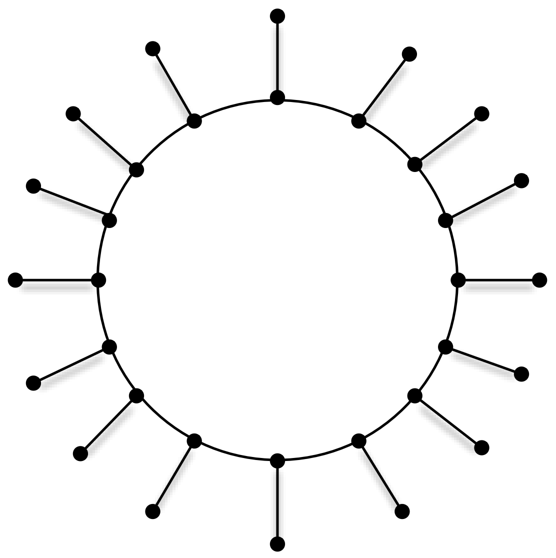


Figure 3.5: A graph solved by Canonical Cleaner in $\mathcal{O}(n)$, because *diameter*$(G) \in \mathcal{O}(n)$

Theorem 9 states that the first visit and visit time is bounded by *diameter*$(G) + \mathcal{O}((n/k)\log n) = \mathcal{O}(\log n + (n/k)\log n) = \mathcal{O}((n/k)\log n)$. From Lemma 15 it follows for $A = V$ that $t_f^* \geq n/k - 1$ and $t_v^* \geq n/k$. This implies a competitive ratio of $\mathcal{O}(\log n)$ for $k \leq n$. If $t_f^* > 0$ it also holds for $k \geq n$. In the case of $t_f^* = 0$, the robots already cover all nodes and every algorithm is optimal for the first visit time. □

For graphs with larger diameter, e.g. *diameter*$(G) \in \mathcal{O}(n)$, the Canonical Cleaner with *starting-node* and *waiting-time* defined as in Theorem 9, we can give an example of its bad performance of $\mathcal{O}(n)$ for the first visit time:

If a graph G consists of $v_1, \ldots, v_{n/2} \in V$ nodes on a circle and to each is exactly one node connected $v_{n/2+1}, \ldots, v_n \in V$, for $k = n/2$ and $s_i = v_i \;\; i = 1, \ldots, n/2$ (see Figure 3.5). With extremely high probability of

$$p \geq 1 - 2^{-k}$$

at least one of the nodes $v_m, m \in \{n/2+1, \ldots, n\}$ will not be chosen as *cycle-start-node*. Since v_m is not on the path from any starting node s_i to the cycle-start-node(s_i). Therefore, it cannot be visited by any robot before the end of the *waiting-time*.

Transforming First Visit to Visit

Another interesting technique is to transform a probabilistic first visit time strategy into a visit time algorithm succeeding with high probability. The only drawback is, that the first visit time and the visit probability for all nodes must be known.

Lemma 16 *Assume there exists a parallel unaware cleaner algorithm $\mathcal{A}$ for k robots on a graph with n nodes, where for all nodes u the probability that the first visit time is less or equal than t_f is at least $p > 0$. Furthermore, t_f and p are known. Then, this cleaning algorithm can be transformed into a canonical algorithm having visit time $\mathcal{O}(\frac{1}{p} t_f \log n)$ with high probability.*

Proof: Let $P(r)$ with $|P(r)| \leq t_f$ be the resulting path of robot r performing algorithm $\mathcal{A}$. Then, the *cycle-start-node* of the canonical algorithm is defined by choosing a random uniform node v_s from $P(r)$. We set *waiting-time*(r)=0.

We now show that this algorithm fulfills the time behavior.

1. The first visit time can be proven as follows:
   Each node is visited with a probability of at least $\frac{p}{t_f}$. However, there are dependencies between these events, since nodes might be visited by the same robot. So, we consider the subpath before a node v of length $\frac{2ct_f \ln n}{p}$ on a cycle C of length $2n$ with $V(C) = V$. Then, at least $c \ln n$ different robots have positive probabilities to visit this interval. Let $1, \ldots, k$ be these robots and let p_i be the probability that one of these robots visits this interval. For these probabilities we have $\sum_{i=1}^{k} p_i \geq \frac{p}{t_f} \frac{ct_f \ln n}{p} = c \ln n$, since otherwise a node exists which is visited with a smaller probability than $\frac{p}{t_f}$.
   The probability for not visiting this interval is therefore
   $$\prod_{i=1}^{k} (1 - p_i) \leq \prod_{i=1}^{k} \exp(-p_i) \leq \exp\left(-\sum_{i=1}^{k} p_i\right) \leq \exp(-c \ln n) \leq n^{-c} .$$
   Since with high probability a cycle-start-node is chosen on the cycle P at most $(2ct_f \ln n)/p$ nodes before v, v will be visited after $t_f + 2\frac{c}{p} t_f \ln n$ steps for the first time w.h.p. From the union bound the claim follows.
2. The visit time follows by the following observation: From the observations above we know that the subpath of length $2ct_f \ln n$ on P before and after any node is visited within time t_f. Therefore, the visit time of a node is at most $4ct_f \ln n + 2t_f$.

$\square$

3.2 The Torus and the Grid Graph

For graphs with larger diameters, other algorithms are necessary. Therefore, we consider torus and grid graphs, where we present optimal unaware cleaner strategies.

Define a $m \times m$-Torus $G_T = (V, E_T)$ graph by $V = [0, \dots, m-1] \times [0, \dots, m-1]$ and with edges $\{(i,j), (i+1 \bmod m, j)\}$ and $\{(i,j), (i, j+1 \bmod m)\}$ for $(i,j) \in V$. Every node has four neighbors, where we call the directions *north*, *east*, *south*, and *west* in the standard way. Parallel unaware robots can clean the torus graph with only a small overhead.

Algorithm 7: Competitive torus cleaner strategy for robot r

```
$(x, y) \leftarrow (s_{r.x}, s_{r.y})$ starting position
for $i \leftarrow 1, 2, \dots, \sqrt{n}$ do
    if random event occurs with probability $(x - s_{r.x} + 1)/(i+1)$ then
        $x \leftarrow x + 1$
    else
        $y \leftarrow y + 1$
    end
    Move to $(x, y)$
end
$H :=$ cycle of Figure 3.7.
while true do
    Move to the next node of $H$
end
```

The first technique, the for-loop of Algorithm 7, is that the cleaner uses a probabilistic process to create a uniform probability distribution over a linear growing and moving set of diagonal nodes. A pure random walk would create a binomial distribution. So, the probability distribution “pushes” to the corners, see Figure 3.6.

Likewise in the canonical algorithm we switch after some time to a deterministic cycling algorithm. The difference is, that this cycle is adapted to the first phase and is a perfect Hamiltonian cycle, see Figure 3.7.

The proof relies on the bait-and-switch-strategy, where the bait is a diagonal field of length t and width $2t_f$. In the neighborhood of such a field at least $\Omega(t)$ robots must be placed at the beginning or the offline strategy does not succeed within first visit time t_f. The first phase of the cleaner strategy moves these robots to a given target node with probability $\mathcal{O}(1/t)$. So, a constant number of robots pass any target node within any time frame of length $\mathcal{O}(t_f)$. Since, the robots' random decisions are independent, an increase of a factor of $\mathcal{O}(\log n)$ gives the time bound for the first phase.

For the second cycling phase, we have chosen the cycle with respect to the first phase, such that the cycle does not destroy the distribution we created in the first

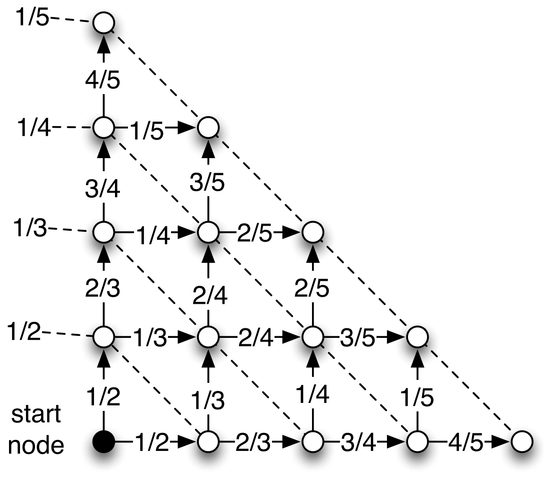


Figure 3.6: Torus cleaner strategy

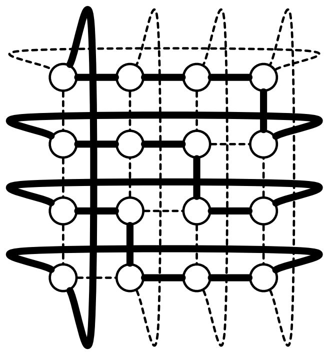


Figure 3.7: Final cycle through the torus

phase. Therefore, the same argument can be reused in order to estimate the maximum distance between two nodes on this cycle.

Theorem 10 *Algorithm 7 is a high probability $\mathcal{O}(\log n)$-competitive visit algorithm for the $m \times m$-torus graph.*

Proof: The following Lemma shows that the torus algorithm distributes the robots with equal probabilities.

Lemma 17 *For all $t \in \{1, \dots, \sqrt{n}\}$, $i \in \{0, \dots, t\}$ the probability that a robot starting at node $(s_{r.x}, s_{r.y})$ is at node $(s_{r.x} + i, s_{r.y} + (t - i))$ after t rounds is $1/(t + 1)$.*

Proof: This follows by induction. For $t = 0$ the probability is 1 that the robot is at the start node $(s_{r.x}, s_{r.y})$. Assume that at round $t - 1$ the claim is true.

For the induction we have to consider three cases:

- If $x = s_{r.x}$ and $y = s_{r.y} + t$, then the probability to move to this point is the product of the stay probability at $(x, y - 1)$ and the probability to increment y. By induction this is $\frac{1}{t}\left(1 - \frac{1}{t+1}\right) = \frac{1}{t+1}$.
- If $y = s_{r.y}$ and $x = s_{r.x} + t$, then the probability to move to this point is the product of the stay probability at $(x, y - 1)$ and the probability to increment x. By induction, this is again $\frac{1}{t}\left(1 - \frac{1}{t+1}\right) = \frac{1}{t+1}$.
- For all other cases, we have to combine the probability to increment x and y, the sum of which is $\frac{t}{t+1}$. By induction we get as probability $\frac{1}{t}\frac{t}{t+1} = \frac{1}{t+1}$ claim follows.

□

Assume that t_f is the optimal offline first visit time for a robot placement in the torus. For the cleaning of a target node (x, y) we choose a set of nodes S with $t - 4t_f$

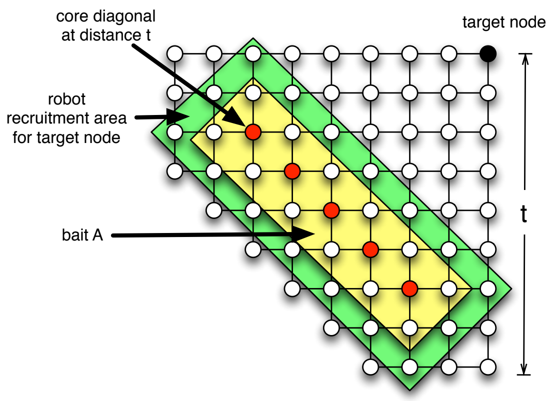

core diagonal at distance t
target node
robot recruitment area for target node
bait A
t

Figure 3.8: The robot recruitment area for robots exploring the target node.

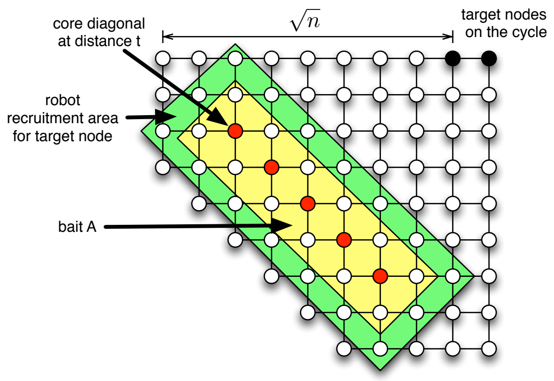

core diagonal at distance t
$\sqrt{n}$
target nodes on the cycle
robot recruitment area for target node
bait A

Figure 3.9: The robot recruitment area for robots on the cycle.

nodes at a diagonal in distance t, see Figure 3.8. $A = N_{t_f}(S)$ is now the bait, i.e. the area, which guarantees the minimum number of robots the recruitment area $N_{t_f}(A)$. Lemma 15 states that at least $|A|/(t_f + 1)$ robots must be in this recruitment area $N_{t_f}(A)$. Now, the cleaning algorithm makes sure that all these robots pass through the target node during the time interval $[t - 2t_f, t + 2t_f]$ with a probability of at least $1/(t + 2t_f + 1)$. Now, the size of $|A|$ is at least $2t_f(t - 4t_f)$. So, the expected number of robots passing through the target node is at least

$$\frac{|A|}{(t_f + 1)(t + 2t_f + 1)} \geq \frac{2t_f(t - 4t_f)(t + 2t_f + 1)}{t_f + 1} \geq \frac{t - 4t_f}{t + 2t_f + 1} .$$

So for $t \geq 10t_f$ we expect at least a constant number of $\frac{1}{2}$ robots passing through any node in a time interval of length $3t_f$. If we increase the time interval to the size of some $ct_f \log n$ for some appropriately chosen constant c, applying a Chernoff bound ensures us to visit this node with at least one robot with high probability.

This proves that in the first phase of the algorithm we visit (and revisit) each node in every time intervals of length $\mathcal{O}(t_f \log n)$.

It remains to be shown that in the second phase, where the algorithm enters the cycle, the distance on the cycle is bounded by $\mathcal{O}(t_f \log n)$. For this, we consider $4t_f < \sqrt{n}$ consecutive nodes on the cycle, which lie on $4t_f$ consecutive diagonals, see Figure 3.9. So, all of the $|A|/(t_f + 1)$ robots in the recruitment area have a target node, which can be reached after $\sqrt{n}$ steps. For each of these target nodes, the probability to be reached by a robot on the corresponding diagonal is at least $\frac{1}{\sqrt{n}}$. The minimum size of $|A|$ is at least $\sqrt{n} - 2t_\theta$, which results in an expected number of at least

$$\frac{2t_f(\sqrt{n} - 2t_f)}{(2t_f + 1)\sqrt{n}} \geq 1 - \frac{t_f}{\sqrt{n}}$$

robots on the target nodes of the cycle. For $t_f \leq \frac{1}{2}\sqrt{n}$ this means that the expected number of robots in an interval of length $4t_f$ is at least $\frac{1}{2}$. Accordingly, the longest empty interval has length of at most $\mathcal{O}(t_f \log n)$ by applying Chernoff bounds on $\mathcal{O}(\log n)$ neighbored intervals.

For $t_f \geq \frac{1}{2}\sqrt{n}$, we consider $\sqrt{n}$ consecutive nodes on consecutive diagonals. Every robot ends the first phase and starts the cycle within this interval with probability $\frac{1}{\sqrt{n}}$. The minimum number of robots to explore all n nodes is at least $\frac{n}{t_f+1}$, which follows by Lemma 15 for $A = V$. Now, for $c\frac{t_f}{\sqrt{n}} \log n$ neighbored intervals on the cycle each of length $\sqrt{n}$ the probability that a single robot chooses a node in this interval is at least

$$\frac{t_f}{\sqrt{n}} \frac{c \log n}{\sqrt{n}} = c\frac{t_f}{n} \log n \ .$$

So, the expected number of robots is $c\frac{n}{t_f}\frac{t_f}{n} \log n = c \log n$ for an time interval of length $c\frac{t_f}{\sqrt{n}}\sqrt{n} \log n = ct_f \log n$. Now, by Chernoff bounds the probability that we find this interval to be empty is at most $n^{-c'}$ for some constants c, c'.

Thus, the maximum distance of two robots on a cycle in the first and second phase is at most $\mathcal{O}(t_f \log n)$ with high probability. Since the visit time is at least the first visit time the competitive ratio of $\mathcal{O}(\log n)$ follows. □

This algorithm can be easily adapted for the grid graph, which consists of the same node set, but edges $\{(i, j), (i + 1, j)\}$ for $i \neq m, (i, j) \in V$ and $\{(i, j), (i, j + 1)\}$ for $j \neq m, (i, j) \in V$.

Theorem 11 *There exists a high probability $\mathcal{O}(\log n)$-competitive visit time cleaning algorithm for the $m \times m$-grid graph with $n = m^2$ nodes.*

Proof: We embed a $2m \times 2m$-torus graph G_T on the $m\times$-grid graph G_G by mapping the four vertices $(x, y), (2m - x + 1, y), (x, 2m - y + 1), (2m - x + 1, 2m - y + 1)$ onto the vertex $(x, y) \in V(G_G)$. Note that the edges of the torus map to edges in the grid.

At the beginning, we choose for each robot a random representative vertex in the torus graph and then we follow the algorithm for the torus graph. The proof is analogous to the one of the torus graph, except for a constant factor increase of the competitive factor. □

3.3 Unaware Parallel Traversal of General Graphs

For general graphs, we use a partition of the graph, which balances the work load of the robots. For the randomized partition, we are inspired by the techniques of embedding tree metrics for graphs [FRT03].

We partition the graph into disjoint recruitment areas $R_1, \dots, R_n \subseteq V$. All robots in a recruitment area R_i have to visit the nodes in a working area W_i which is a proper subset of R_i. These sets are defined by a random process such that each node has a constant probability to be contained in a working area and we show that the number of robots in the recruitment area is large enough to ensure that this node is visited with constant probability. This constant probability will be increased later on by repeating the partitioning several times.

We give a formal description of the sets used in Algorithm 8. The recruitment partition uses center nodes $c_1, \dots, c_n$ which are given by a random permutation π of all nodes $V = \{v_1, \dots, v_n\}$, i.e. $c_i = v_{\pi(i)}$. The partition is based on the neighborhood set $N_\ell(u)$, which is the set of nodes v for which the distance to u is at most ℓ. So, we define for a radius ℓ and for all $i \in \{1, \dots, n\}$.

$$R_i := N_\ell(v_{\pi(i)}) \setminus \bigcup_{j=1}^{i-1} N_\ell(v_{\pi(j)}) \ . \tag{3.1}$$

The working areas are defined for radius ℓ and an estimation of the first visit time $t \in [t_f, 2t_f]$ as

$$U_i \ := \ N_{l-2t}\left(v_{\pi(i)}\right) \setminus \bigcup_{j=1}^{i-1} N_{\ell+2t}\left(v_{\pi(j)}\right) \tag{3.2}$$

$$W_i \ := \ N_t\left(U_i\right) \tag{3.3}$$

We denote by $W = \bigcup_{i=1}^{n} W_i$ the set of nodes that will be worked on and let $U := \bigcup_{i=1}^{n} U_i$.

These definitions are used for a probabilistic cleaning Algorithm 8, which covers a constant part of the graph.

Algorithm 8: ONE-SHOT-CLEANING $G = (V, E)$ using $V = R_1 \dot\cup \cdots \dot\cup R_n$ and $W_1, \dots, W_n \subseteq V$

```
Choose $i$ such that $s_r \in R_i$
$T_i \leftarrow$ STEINER-TREE-APPROXIMATION($W_i$)
$C_i \leftarrow$ DFS-Cycle($T_i$)
Move to a random node of $C_i$
Walk on $C_i$ for $68t \log n$ rounds
Move back to $s_r$
```

The One-shot-cleaning algorithm makes use of a straight-forward constant factor Steiner-tree approximation based on Prim's minimum spanning tree algorithm[Pri57], presented as Algorithm 9.

Algorithm 9: Steiner-Tree-Approximation with input $G = (V, E)$, $W \subseteq V$

```
$(C_1, \ldots, C_p) \leftarrow$ connected components of $W$ in $G$
while $p > 1$ do
    Choose the component $C_j$ with the nearest node to $C_1$
    $W \leftarrow W \cup$ (node set of shortest path between $C_1$ and $C_j$ to $W$)
    $(C_1, \ldots, C_p) \leftarrow$ connected components of $W$
end
return spanning tree of $C_1$
```

The following lemma shows that every node is chosen with a probability of at least $\frac{1}{4}$ to be the target of a robot cleaning in some area W_i.

Lemma 18 *For a graph G, a node $v \in V$, β chosen randomly from $[1, 2]$, a random permutation π over $\{1, \ldots, n\}$, and for $l = 8\beta t \log n$ the probability that $v \in W$ is at least $\frac{1}{4}$.*

Proof: We will prove that $P(v \in U) \geq \frac{1}{4}$, which implies the claim because $U \subset W$.

Consider the first node w in the $\ell + 2t$-neighborhood of v according to the random permutation π, i.e. $w = u_{\pi(i*)}$ where $i^* = \min\{i \mid |v, u_{\pi(i)}| \leq \ell + 2t\}$. If w is closer than $\ell - 2t$ to v, i.e. $|v, w| \leq \ell - 2t$, then v is in the working area of w (and U), since no node with a smaller index can be closer than w, i.e. $w \in U_{i*} \subseteq U$. On the other hand if this node is in the critical distance $|v, w| \in (\ell - 2t, \ell + 2t]$, then it is excluded from U_{i*} and since i^* has the smallest index in the vicinity it is also not in any other working area, i.e. $v \notin U$. Since π is a random permutation the probability of $v \in W$ is given by the number of elements in the closer vicinity:

$$P_\ell(v \in U) = \frac{|N_{\ell-2t}(v)|}{|N_{\ell+2t}(v)|}$$

This implies

$$\prod_{i=0}^{2\log n} P_{\ell+4it}(v \in U) = \frac{|N_{\ell-t}(v)|}{|N_{\ell+8t\log n+2t}(v)|} \geq \frac{1}{n} \tag{3.4}$$

Now, we choose β randomly from $\{1, 1 + \frac{1}{2\log n}, 1 + \frac{2}{2\log n}, \ldots, 1 + \frac{2\log n - 1}{2\log n}\}$ and compute $\ell = 8\beta t \log n$. Hence,

$$P(v \in U) = \frac{1}{2\log n} \sum_{i=0}^{2\log n - 1} P_{8t\log n + 4it}(v \in W)$$

Assume that $P(v \in U) < \frac{1}{4}$, then at least half of all values of $(P_{8t\log n+4it}(v \in W))_{i\in\{0,\ldots,2\log n-1\}}$ are smaller than $\frac{1}{2}$. Then, we observe the following.

$$\prod_{i=0}^{2\log n} P_{8t\log n+4it}(v \in U) < \left(\frac{1}{2}\right)^{\log n} = \frac{1}{n} \ ,$$

which contradicts (3.4). Therefore $P(v \in W) \geq P(v \in U) \geq \frac{1}{4}$.

The same argument holds, if we choose β randomly from the real interval $[1, 2]$.

□

Now, we investigate whether there are enough robots in the recruitment area R_i in order to explore W_i. The number is large enough if a given node is explored with a constant probability. However, there is a major problem: U_i, W_i, or R_i might be disconnected. Robots might travel long routes between the nodes in W_i outside of W_i or even R_i.

Therefore, we need an upper bound on the size of these connecting routes. This has been the motivation to extend U with a surrounding of t neighborhood nodes. So, for $\beta \in [1, 2]$ we have the following lemma.

Lemma 19 *For $\ell = 8\beta t \log n$, let T_i be the tree connecting all nodes in W_i constructed in Algorithm 9. Then,*

$$|V(T_i)| \leq 17|W_i| \log n \ .$$

Proof: Each of the p connected components $C_1, \ldots, C_p$ of W_i has at least one node of U and its t-neighborhood. So, C_j has at least t nodes, implying $|W_i| \geq pt$. Every node of W_i has a distance of at most $\ell = 8\beta t \log n$ to $v_{\pi(i)}$. The maximum distance between two components is thus at most $16\beta t \log n$ because of the triangle inequality. Which implies that at most $16(p-1)\beta t \log n$ nodes are added to connect the original p connected components. So,

$$\begin{aligned} |V(T_i)| &\leq 16(p-1)\beta t \log n + |W_i| \\ &\leq 16 \ \frac{p-1}{p} \ |W_i| \log n + |W_i| \\ &\leq 17|W_i| \log n \ . \end{aligned}$$

□

The following lemma shows that the ONE-SHOT-CLEANING algorithm needs only a logarithmic overhead.

Lemma 20 *The number of moves of a robot using* ONE-SHOT-CLEANING *for $\ell = 8\beta t \log n$ and $\beta \in [1, 2]$ is at most $100t \log n$.*

Proof: The maximum distance of any node from u to W_i is at most $\ell - t = 8\beta t \log n - t \leq 16t \log n$. So, moving to the start node and moving back to the start node needs at most $32t \log n$ rounds. Moving on C_i needs $68t \log n$ rounds resulting in $100t \log n$ rounds. □

Now, we need to show that the number of robots in the recruitment area R_i is large enough. This follows by Lemma 15 substituting $A = W_i$.

Lemma 21 *If the robots are placed such that a first visit time of t_f is possible for the offline algorithm, and $t \in [t_f, 2t_f]$, then for the number k_i of robots originally placed in R_i we have*

$$k_i \geq \frac{|W_i|}{t_f + 1} \geq \frac{|W_i|}{2t} .$$

Proof: A single robot can explore at most $t_f + 1$ nodes in the first t_f rounds. Therefore the minimum amount of nodes to be explored by all robots in R_i is $k_i(t_f + 1) \leq 2k_i t_f$. □

These observations allow us to find a general strategy for the first visit problem for unaware parallel cleaners.

Algorithm 10: High probability first visit cleaner of $G = (V, E)$

```
for $i \in \{1, 2, \ldots, \log n\}$ do
    $t \leftarrow 2^i$
    for $j \in \{1, \ldots, 4(c + 1) \ln n\}$ do
        Choose randomly $\beta \in [1, 2]$
        Choose random permutation $\pi$ over $V$
        One-shot-cleaning$(G, \ell = 8\beta t \log n, t, \pi)$
    end
end
```

Theorem 12 *Algorithm 10 is a high probability $\mathcal{O}(\log^2 n)$-competitive first visit algorithm for every undirected graph.*

Repeating the one-shot-cleaning $\mathcal{O}(\log n)$ times gives us a high probability.

Proof: Consider the round of the outer loop, where $t = 2^i \in [t_f, 2t_f]$, where t_f is the first visit time of the optimal algorithm. We show that in this round all nodes will be explored with high probability. Lemma 20 states that the number of robot moves of one-shot-cleaning is bounded by $100 \cdot 2^i \log n$. So, the overall number of each robot moves is bounded by $800(c + 1) \log^2 n$.

For any node u the probability, that the one-shot-cleaning algorithm for $\ell = 8\beta t \log n$ chooses $u \in W$ is at least $\frac{1}{4}$ following Lemma 18. If u resides in W_i, the number of robots performing the cleaning is at least $|W_i|/(2t)$ implied by Lemma 21. These k_i robots have to explore a cycle of length at most twice the size of the connected Steiner-tree computed in Algorithm 9. These are at most $34|W_i| \log n$ nodes. Now, Algorithm 8 starts with a random node node and explores $68t \log n$ nodes. So, after one execution of the ONE-SHOT-CLEANING algorithm the probability of a node not to be explored is at most

$$1 - \frac{1}{4} \frac{68t \log n}{34|W_i| \log n} = 1 - \frac{t}{2|W_i|}$$

The cleaning is independently executed for each of the k_i robots $k_i \geq \frac{|W_i|}{2t}$ times.

$$\left(1 - \frac{t}{2|W_i|}\right)^{\frac{|W_i|}{2t}} \leq e^{-\frac{1}{4}}$$

Hence, the maximum probability of a node not to be explored after $4(c+1)\ln n$ repetitions is at most $\frac{1}{n^c}$. □

The visit time problem needs more moves, since a robot may make a fast first visit, but does not know when to end. Our solution is to guess the first visit time.

Algorithm 11: High probability visit of $G = (V, E)$

```
Choose uniform at random $i \in \{1, 2, \ldots, \log n\}$
$t \leftarrow 2^i$
Choose randomly $\beta \in [1, 2]$
Choose random permutation $\pi$ over $V$
ONE-SHOT-CLEANING$(G, t, \beta, \pi)$
Traverse the graph by DFS yielding a cycle $C$ with $V(C) = V$ of length $2n$
Go to a random node visited during the one shot cleaning
while true do
    Walk to the next node of $C$
end
```

Theorem 13 *Algorithm 11 is an high probability $\mathcal{O}(\log^3 n)$-competitive visit algorithm for every undirected graph.*

Proof: Lemma 18 implies that $P(w \in W_i) \geq \frac{1}{4}$ if $\ell = 8\beta t \log n$. The probability that a robot chooses the correct value $t = 2^i \in [t_f, 2t_f]$ is $1/\log n$. So, the probability that a node is visited within first visit time $800ct_f \log n$ is at least $p = \frac{1}{4 \log n}$. By Lemma 16 this implies a visit time algorithm with high probability with time $\mathcal{O}(t_f \log^3 n)$.

□

3.4 Empirical Evaluation of Parallel Unaware Cleaning

In this section, we will take a look at simulations of cleaning algorithms shown in this chapter. Algorithms provided in this chapter have crude approximations in them. These may be unimportant for the proofs as they only change constants for the run-time, but for any experiment constant factors of $40c$ or even $800c$ are problematic. Preliminary experiments in the Master Thesis of Felix Thein [The15] indicate not only that the approximation of $800c$ is far too crude and is not applicable for simple graphs. The constant factors may appear in the proof, but they do not appear in the measurements. For example, the factor of 34 by the MST approximation of the Steiner-tree will not apply to the empirical evaluation on a grid graph. On the other hand a factor of 2 from using a DFS path instead of a Hamiltonian path, turns out to be a huge disadvantage when comparing with other algorithms. A circle that uses a DFS path becomes similar to the line with the same problems for border nodes and also has the full factor of 2 larger paths between those nodes compared to the, in this case easy to compute, Hamiltonian cycle.

We here compare four different algorithms on a torus graph.

- CC - The Canonical Cleaner (Algorithm 6) using a randomly chosen *cycle-start-node* and diameter minus distance traveled as *waiting time* (the same was used for Theorem 9). To improve the comparison to the preliminary experiments we use a Hamiltonian cycle instead of a DFS cycle to traverse the graph.
- CR - The Conscientious Reactive Algorithm provided as Algorithm 5.
- HPV - The High Probability Visit Algorithm (Algorithm 10). This algorithm has by far the largest constants in the proof. Again, we replace the final DFS cycle over the whole graph with an Hamiltonian cycle for traversal.
- TC - The Torus Cleaner (Algorithm 7) is used unchanged.

To compare these algorithms we do not simply provide a measurement of the first visit time of the graph and the visit time. To better characterize the cleaning of a graph, we want to provide measurements depending on time. Therefore we characterize the cleaning with the concept of *idleness*, which is defined as time since the node has been visited last. This concept is similar to our visit time, but because of its dependence on time of measurement better fit to describe an experiment.

The idleness of node v_i (*instantaneous node idleness*) at time t is described by :

$$Idl_t(v_i) = t - t_{last\ visit\ of\ v_i}$$

The idleness of the whole graph G (*average instantaneous node idleness*) at any time t:

$$Idlavg_t(G) = \sum_{i=1}^{n} Idl_t(v_i)/n$$

We also provide (*maximum instantaneous node idleness*) which allows us to directly see visit and first visit time for the cleaning.

$$Idlmax_t(G) = \max_{i=1,\ldots,n}(Idl_t(v_i))$$

The first visit time can be determined by taking a look at the first extremum of the curve and the visit time by determining the maximum of the curve over t. As a third benchmark to directly see how the first visit progresses, we provide graphs for the ratio of nodes visited.

For all the following experiments, we do 11 simulations for each setting. Then we present the run with the median first visit time. The duration of each algorithms simulation is exactly twice as long as it takes an algorithm to visit every node $(2t_f(\mathcal{A}))$. This is not enough for HPV and CR to converge to a steady state. The torus is set to have side length $m = 160$ resulting in a graph of size $n = 25600$ nodes.

Same Node Placement with few Robots

For the first setting, we fix the number of robots to $k = m = 160$ and place them all on the same starting node. This is kind of a simple scenario. As t_f is larger than the diameter of the graph, we expect algorithms such as the Canonical Cleaner to do very well. In Figure 3.10 we can see that the average node idleness behaves best for the Canonical Cleaner. Its rather even distribution allows the CC Algorithm to be less than a factor of 2 larger than a theoretical optimal offline solution which could achieve $Idlavg_t(G) = 80$ by distributing robots perfectly even (see Figure 3.10). The $Idlmax_t(G)$ is for all algorithms dominated by the first visit time. Here we can also see a larger difference to an optimal offline algorithm, which would be able to explore this scenario in no more than $\lceil m^2/k \rceil = m < t_f \leq 3m/2$ and therefore in at most $Idlmax_t(G) \leq 240$. Thus the Canonical Cleaner and Torus Cleaner algorithms, which solve this scenario best, are at least a factor of 4 worse than an optimal solution (see Figure 3.11 and Table 3.1).

Table 3.1: Exploration Times

	1	2	3	4	5	6	7	8	9	10	11
CC	764	783	957	976	1031	1055	1091	1095	1114	1227	1501
CR	2601	2647	2654	2685	2734	2756	2787	2903	2997	3080	3235
HPV	1897	2063	2139	2141	2181	2214	2249	2536	2588	3312	3337
TC	799	959	959	960	1106	1119	1120	1120	1277	1280	1434

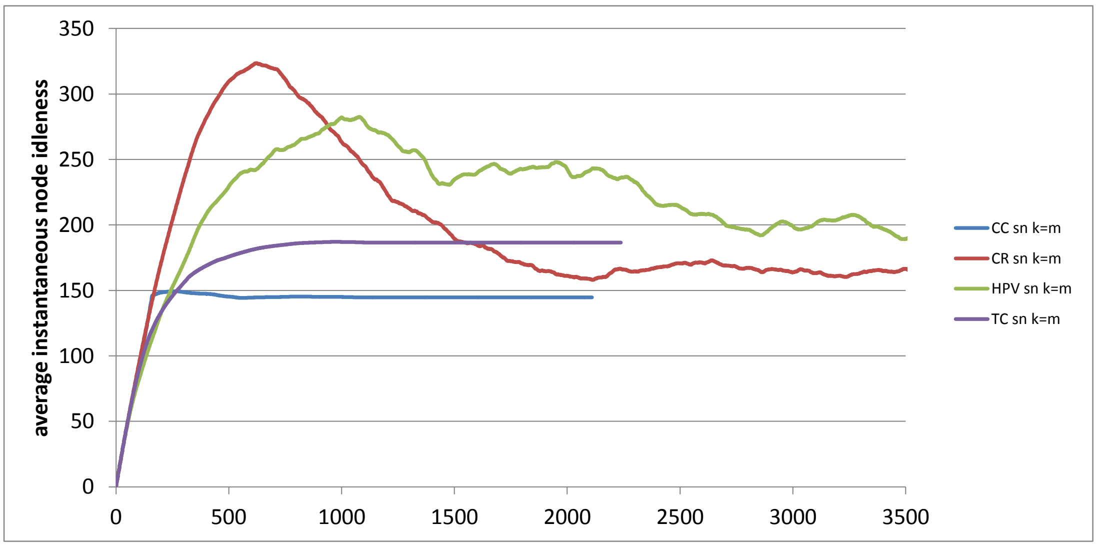


Figure 3.10: Same Node placement m Robots

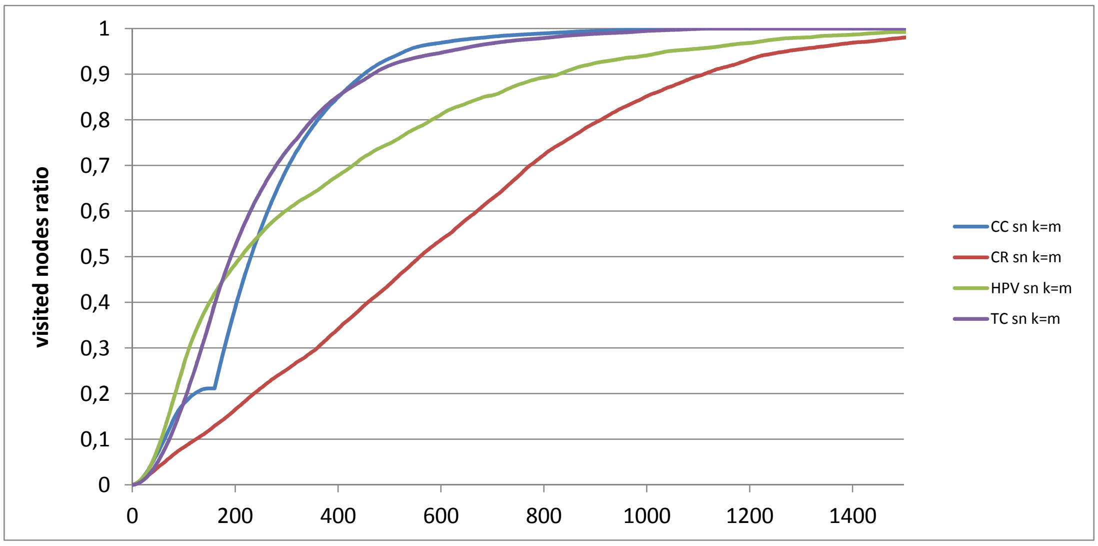


Figure 3.11: Same Node placement m Robots

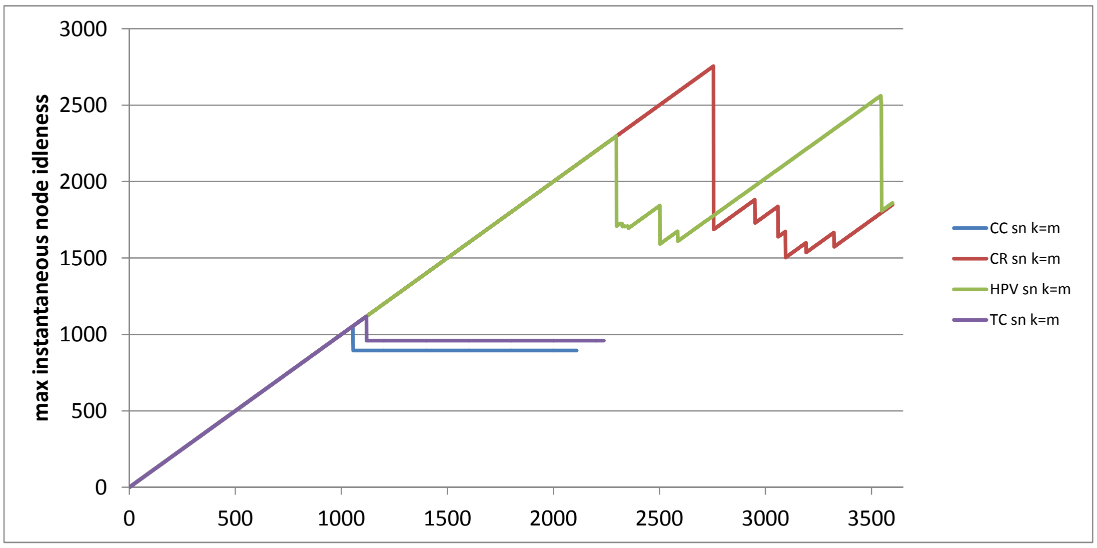


Figure 3.12: Same Node placement m Robots

Same Node Placement with many robots

For the next run of experiments, we increase the amount of robots by a factor of 10. This allows the optimal offline algorithm to achieve a first visit time of $t_f = m - 1$. So the first visit is now determined by the diameter. In Table 3.2 we can see that the Torus Cleaner reliably hits all nodes by following its random distribution (Figure 3.6). But is still a factor of 2 off from an optimal algorithm. We get this factor from only walking in one direction (north and east) instead of choosing the direction randomly. The CR algorithm looses ground in this scenario as its random process for choosing nodes hinders dispersing robots quickly to the farthest away nodes. We can also see that the TC algorithms average instantaneous node idleness is the largest in the long term, because the TC algorithm will only disperse robots starting at a single node to a diagonal in the grid graph.

Table 3.2: Exploration Times

	1	2	3	4	5	6	7	8	9	10	11
CC	261	265	267	271	274	278	281	286	293	301	306
CR	1072	1088	1143	1144	1164	1200	1216	1221	1239	1263	1294
HPV	281	298	300	302	334	337	346	349	408	474	501
TC	318	318	318	318	318	318	318	318	318	318	318

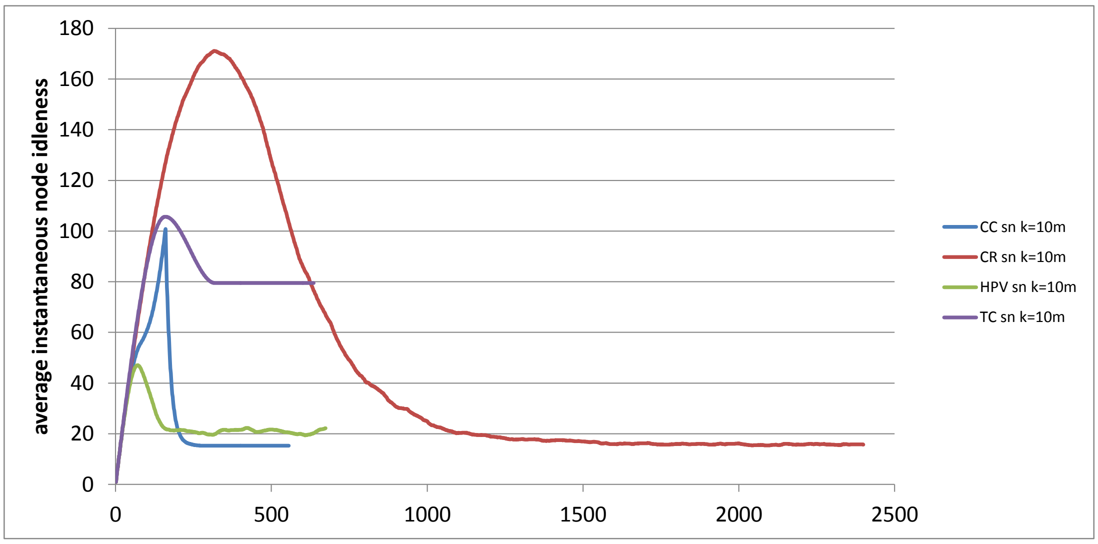


Figure 3.13: Same Node placement $10m$ Robots

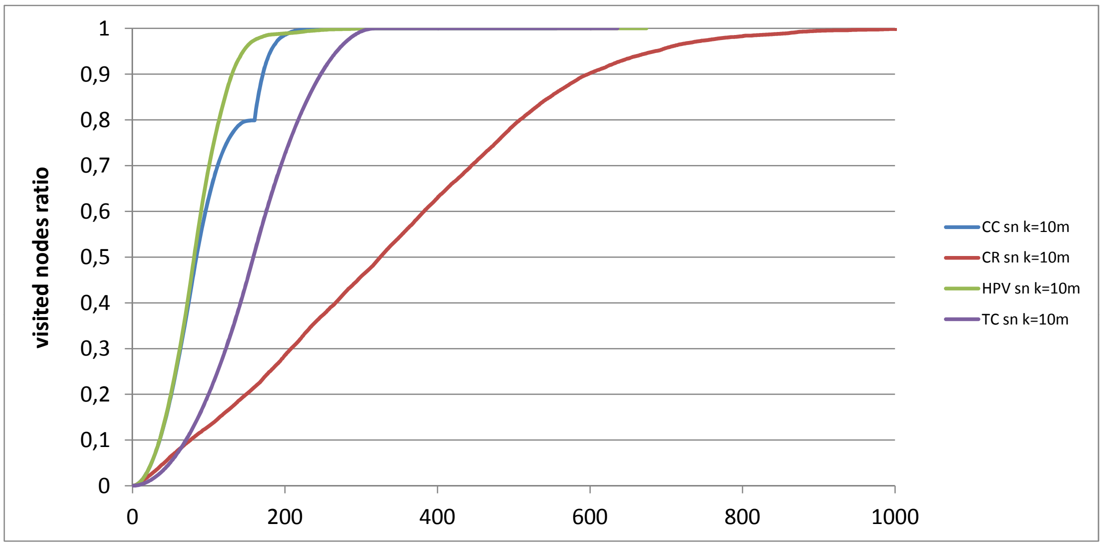


Figure 3.14: Same Node placement 10m Robots

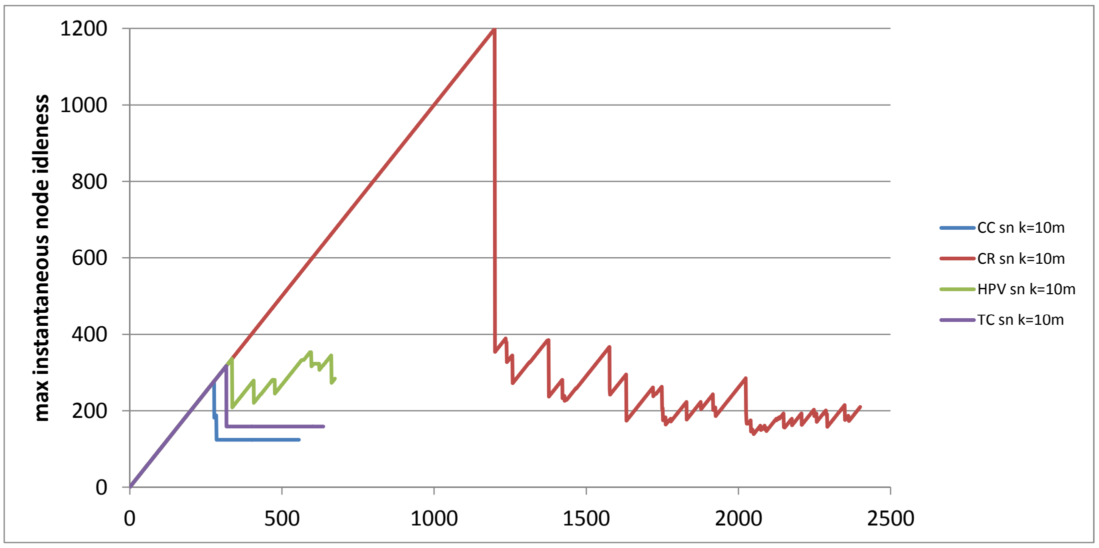


Figure 3.15: Same Node placement 10m Robots

Random Placement with few Robots

In this section we go back to using $m = k = 160$ robots, but set them on k random positions in the graph.

If robots are placed randomly, this benefits algorithms which otherwise fail to disperse robots quickly. Comparing Figure 3.18 to 3.12, we can see that CR has now a clear advantage to the HPV cleaner. Even TC cleaner can profit slightly for the average case, because dispersing robots on a single diagonal in the torus has a negative effect on the node idleness.

Table 3.3: Exploration Times

	1	2	3	4	5	6	7	8	9	10	11
CC	785	786	802	878	962	1028	1093	1229	1273	1322	1962
CR	1451	1469	1632	1640	1726	1800	1826	1828	1930	1960	1996
HPV	1897	2063	2139	2141	2181	2214	2249	2536	2588	3312	3337
TC	770	897	913	926	945	952	965	1042	1107	1200	1265

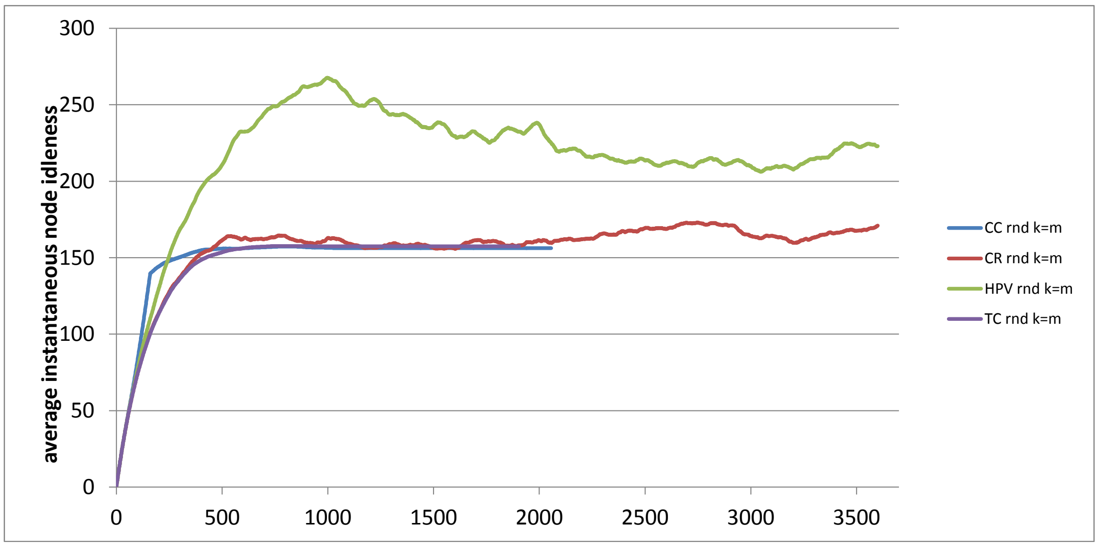


Figure 3.16: Random placement $k = m$ Robots

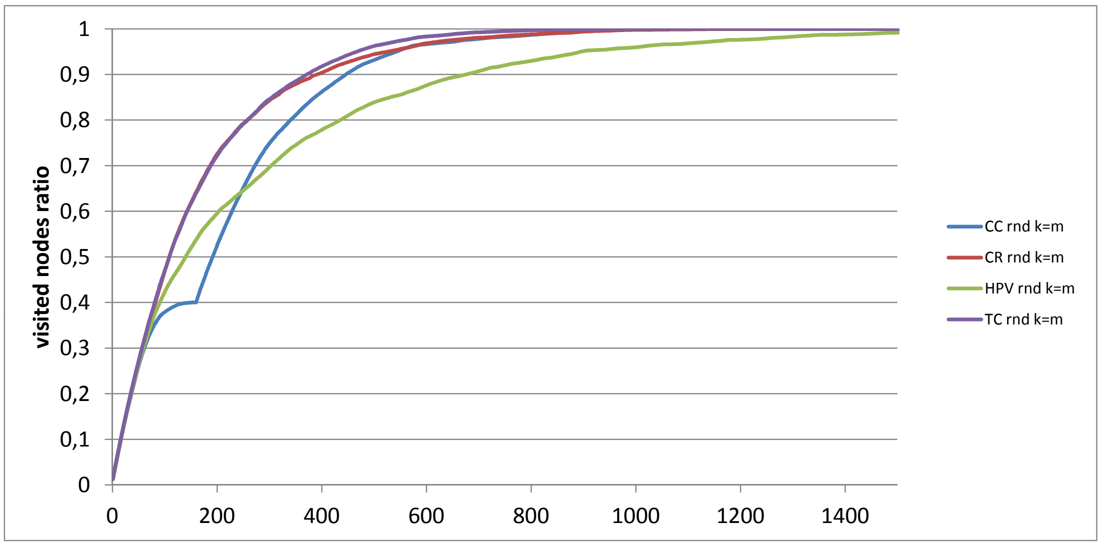


Figure 3.17: Random placement $k = m$ Robots

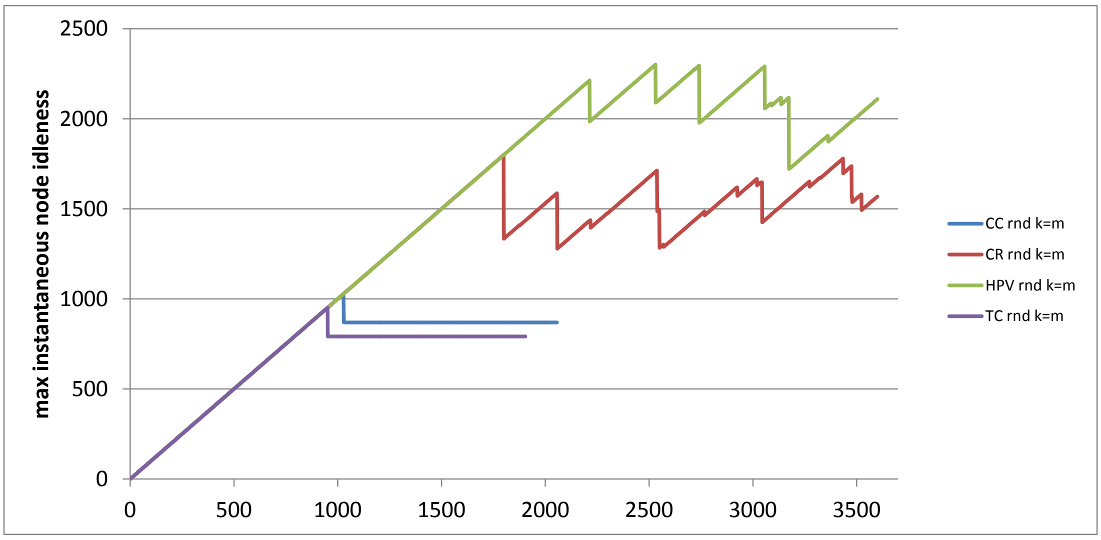


Figure 3.18: Random placement $k = m$ Robots

Random Placement with many Robots

In our last cleaning experiment, we use many robots $k = 1600 = 10m$ and disperse them randomly in the torus. If robots were distributed evenly instead of randomly it would allow for a first visit time of $t_{opt} = t_f = \lceil \frac{n-k}{k} \rceil = 15$. This is improbable. However, there is an averaging effect. While it is hard to calculate what is the expected t_f we can calculate how many robots in our experiment will fall into a square sized $\frac{n}{k} \times \frac{n}{k}$ and we can check what is the probability of such a square being hit by less than half the robots needed to clean it in time $\frac{n}{k}$:

$$P(\le \frac{n}{2k} robots) = \sum_{i=0}^{\lceil \frac{n}{2k} \rceil} \binom{k}{i} * p^i * (1-p)^{k-i}, p = \frac{n}{k^2}$$

For our experiment's parameters, 2 such squares will exist in expectation. However, if we allow robots to come to the square from all nodes in distance $\frac{n}{2k}$. With an overwhelming probability no square with such a vicinity containing less than $\frac{n}{2k}$ robots exist. Therefore we conjecture $t_f \approx 2n/k$ for this scenario.

In Diagram 3.19 we can see why the Canonical Cleaner has problems on graphs with a large diameter. For $diameter(G) = \sqrt{n}$ and small t_f, the average instantaneous node idleness increases, while a rising number of robots stop moving and wait for the rest of the robots to find their random position in the graph. We can not perceive the same kind of problem for the instantaneous max idleness. If the graph didn't have a uniform topology, where the paths to the randomly chosen node can solve the first visit, or a larger diameter, ensuring longer waiting times for the CC algorithm, then the Canonical Cleaner would perform worse (as was shown in Figure 3.5) .

Table 3.4: Exploration Times

	1	2	3	4	5	6	7	8	9	10	11
CC	214	219	221	225	227	234	234	258	258	259	261
CR	151	157	159	163	164	168	184	184	189	210	227
HPV	188	204	214	217	235	239	242	249	282	296	323
TC	144	149	150	160	163	170	170	178	197	199	210

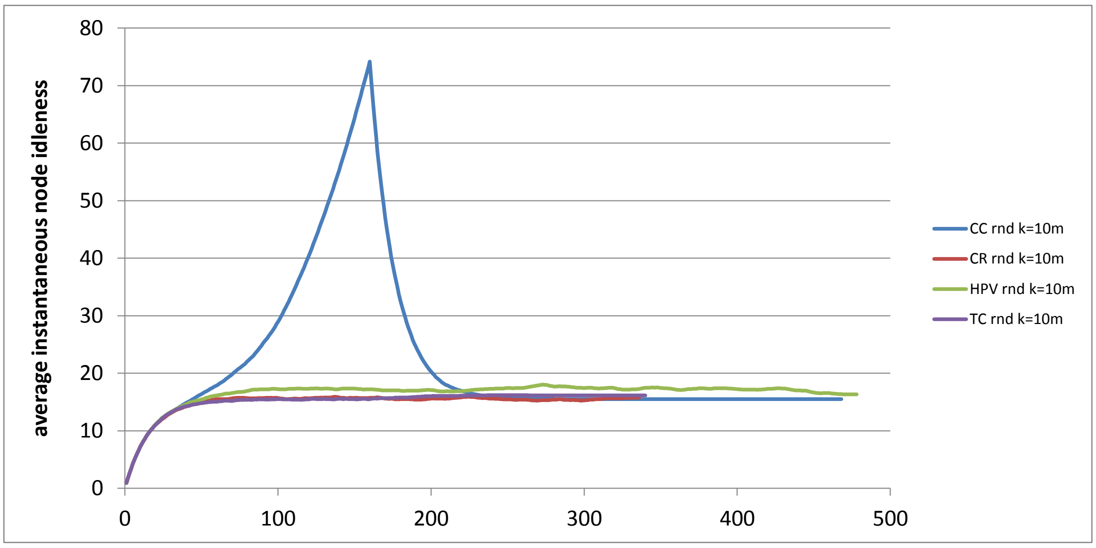


Figure 3.19: Random placement $10m$ Robots

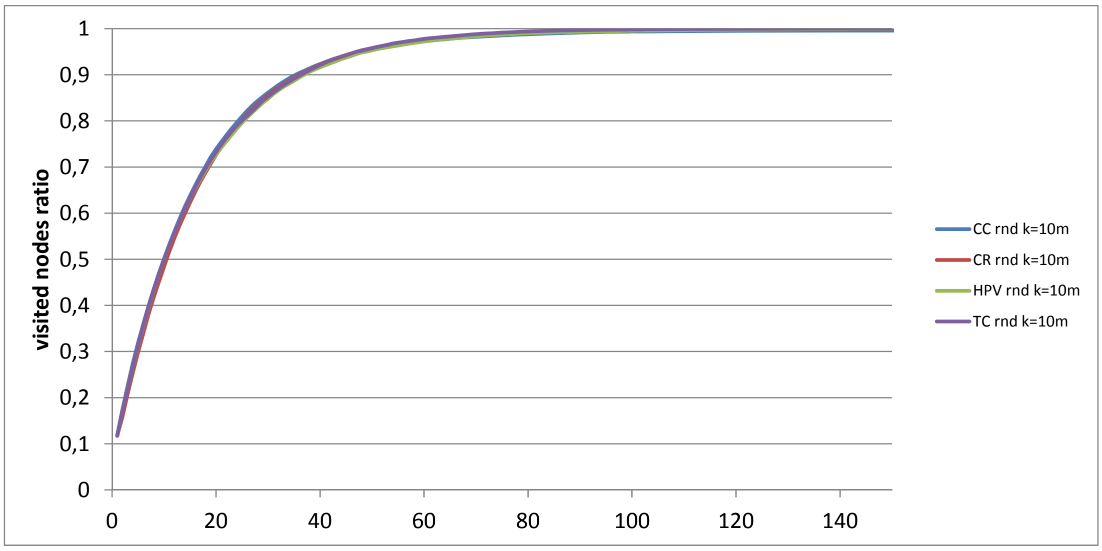


Figure 3.20: Random placement $10m$ Robots

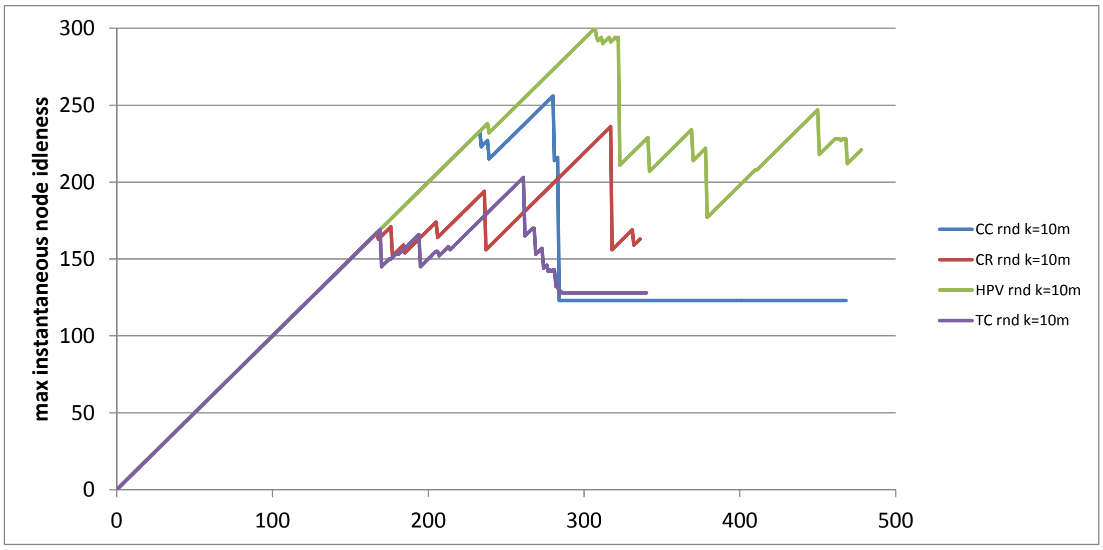


Figure 3.21: Random placement $10m$ Robots

4 Conclusion

In this thesis we discussed the cooperation of robots in online scenarios. The robot teams were either forced to work without a map of their task or without being allowed to be aware of each other.

In the Exploration chapter we studied the exploration with multiple robots of grid graphs and trees without the use of a map. We showed that the grid graph allows us to construct the same competitive lower bound as in trees of $\Omega(\frac{\log k}{\log\log k})$.

Further, we discussed randomized algorithms. For randomized tree exploration the lower bound of $\Omega(\frac{\log k}{\log\log k})$ is equal to the deterministic bound by Dynia et al. [DLS07]. For the grid graph, we were only able to show a weaker lower bound of $\Omega(\frac{\sqrt{\log k}}{\log\log k})$.

On the positive side, the upper bounds for online exploration could be significantly improved in this thesis. We were able to bound competitiveness for the exploration of grid graphs by $\mathcal{O}(\log^2 n)$. Before this work, the best known bound for this scenario was the algorithm provided by Dynia et al. in [DKS06] with competitiveness of $\mathcal{O}(\sqrt{n})$.

For trees, we were able to show the first sub-polynomial competitive bound on trees. This bound is non-trivial for extreme values of k, d, n, but for k and d being small polynomials it can be expressed as $n^{o(1)}$. By improving the so far best competitive bound of $\mathcal{O}(\frac{k}{\log k})$, this closed down the exponential gap between upper and lower bound that had been open for nearly a decade.

In the last section, we showed some empirical results for our exploration algorithms. For the largest graph sizes we can simulate, the Yo* is only slightly faster than a DFS algorithm with a single robot. Positively, with the optimization of not stopping while redistributing robots, we created the Hedge Cutter algorithm which solves the comb graph with $\mathcal{O}(\log d)$ competitiveness and for which we have yet to show the existence of a tree with a worse competitive bound. The simulation of the Hedge Cutter shows very low constant factors. On the largest simulatable instance with 4 million nodes, we are by a factor of 6 slower than an optimal offline solution.

To put these results into perspective we conclude with a list of multi-robot exploration algorithms for different graph types and values for k and d where they are efficient (Table 4.1). We omit the competitive factors above poly-logarithmic and below polynomial in k and d for clarity and refer to the appendix for these.

In the Parallel Unaware Cleaning chapter we discussed a central question of distributed algorithms: How much do we benefit from communication? Or in other

k	d	graph	competitive factor	algorithm
$\log^{c'} n$	any	any	$\mathcal{O}(\log^{c'} n)$	DFS
$\log^{c'} n$	any	tree	$\mathcal{O}(\frac{\log^{c'} n}{\log\log n})$	Greedy [FGKP06]
any	$\mathcal{O}(\sqrt{n})$	rect. obst.	$\mathcal{O}(\log^2 n)$	Algorithm 2
any	$\log^{c'} n$	any	$\mathcal{O}(\log^{c'} n)$	Yo-yo
n^c	n^c	tree	$k^{o(1)}$	Yo*
n^c	n^c	comb	$\mathcal{O}(\log n)$	Hedge Cutter
n	$\mathcal{O}(\sqrt{n})$	rect. obst.	$\mathcal{O}(\log n)$	Algorithm 2
$d \cdot n^{1+c'}$	any	any	$\mathcal{O}(\frac{1}{c'})$	Dereniowski et al. [DDK+13]
n^d	any	any	1	Flooding/Greedy

Table 4.1: Competitive exploration time ratios for constants $0 < c < 1$ and $c' > 0$

words: Can we cope with a parallel problem if communication is not available?

We have shown that a first visit can be achieved with an overhead of $\mathcal{O}(\log^2 n)$ and visit with $\mathcal{O}(\log^3 n)$ in general graphs. This implies that cooperation is not dependent on communication for tasks such as searching or cleaning. Communication only improves such tasks by a poly-logarithmic factor depending on the graph type required for the model.

On the grid and torus, with an even stronger bound of $\mathcal{O}(\log n)$, a non-communicating team can perform best. This matches the lower bound of $\Omega(\log n)$ given by the coupon collector's problem. Unlike the algorithm presented for general graphs the parallel unaware cleaner strategy for torus and grids have provably small constant factors involved. Furthermore, the grid represents a good model for such robots doing vacuuming or mowing work.

The empirical evaluation indicates that the overhead for the Torus Cleaner Algorithm is small enough to make it a viable alternative for any application of patrolling algorithms with expensive communication. In addition, the solution for general graphs (Algorithm 10) is, despite its large constants in the proof, only off by about a factor of 2 from the specialized Torus Cleaner Algorithm. Therefore, it can also be considered a viable algorithm for application.

For both scenarios, we have presented algorithms with proven upper bounds for the competitiveness. Especially for the exploration we improved the best known solution from polynomial to sub-polynomial overhead. With our algorithms, we showed how much additional run-time these algorithms require compared to an optimal offline solution utilizing a map and communication between robots. Our results show, that when forced to give up communication completely or to work without a map, this comes at a time-cost for a team, which has to be considered for designing such a cooperative exploration or cleaning system.

For the future we hope that the algorithms presented here can be further improved, since only the bound for the parallel unaware cleaning of grids is tight. General graphs and trees may still be improvable to the same logarithmic bound.

Moreover, for exploration there is a task left to do. While the Yo*-algorithm is the algorithm with the best shown bounds for trees, the presented Hedge Cutter algorithm lacks any proven bounds on unrestricted trees. But because we can not even generate a counter example that forces it into a bad run-time we are optimistic for the future. We conjecture its run-time to be logarithmic or poly-logarithmic and will try to find a proof for this soon.

Overall during this thesis we improved the state of the art in cooperative graph exploration tremendously. We believe the presented results and techniques will inspire new work and promote new progress on the subject.

Bibliography

[AH00] Susanne Albers and Monika R. Henzinger. Exploring Unknown Environments. *SIAM Journal on Computing*, 29(4):1164, 2000.

[AKS99] Susanne Albers, Klaus Kursawe, and Sven Schuierer. Exploring unknown environments with obstacles. In *Proceedings of the tenth annual ACM-SIAM symposium on Discrete algorithms*, SODA '99, pages 842–843, Philadelphia, PA, USA, 1999. Society for Industrial and Applied Mathematics.

[AKS08] Susanne Albers, K. Kursawe, and S. Schuierer. Exploring unknown environments with obstacles. *Algorithmica*, 32(1):123–143, 2008.

[Aro96] S. Arora. Polynomial time approximation schemes for euclidean tsp and other geometric problems. In *Foundations of Computer Science, 1996. Proceedings., 37th Annual Symposium on*, pages 2–11, Oct 1996.

[ARS$^+$04] Alessandro Almeida, Geber Ramalho, Hugo Santana, Patrícia Tedesco, Talita Menezes, Vincent Corruble, and Yann Chevaleyre. Recent advances on multi-agent patrolling. In *Advances in Artificial Intelligence–SBIA 2004*, pages 474–483. Springer, 2004.

[BBF$^+$96] Piotr Berman, Avrim Blum, Amos Fiat, Howard Karloff, Adi Rosén, and Michael Saks. Randomized robot navigation algorithms. In *Proceedings of the seventh annual ACM-SIAM symposium on Discrete algorithms*, SODA '96, pages 75–84, Philadelphia, PA, USA, 1996. Society for Industrial and Applied Mathematics.

[BCMGX11] P. Brass, F. Cabrera-Mora, A. Gasparri, and Jizhong Xiao. Multirobot tree and graph exploration. *Robotics, IEEE Transactions on*, 27(4):707–717, Aug 2011.

[BCR93] R.A. Baezayates, J.C. Culberson, and G.J.E. Rawlins. Searching in the Plane. *Information and Computation*, 106(2):234 – 252, 1993.

[BEBFY94] Eldad Bar-Eli, Piotr Berman, Amos Fiat, and Peiyuan Yan. Online navigation in a room. *J. Algorithms*, 17:319–341, November 1994.

[Bek06] Tolga Bektas. The multiple traveling salesman problem: an overview of formulations and solution procedures. *Omega*, 34(3):209 – 219, 2006.

[Ben73] CH Bennett. Logical reversibility of computation. *Maxwell's Demon. Entropy, Information, Computing*, pages 197–204, 1973.

[Ben94] Michael A. Bender. The power of team exploration: Two robots can learn unlabeled directed graphs. In *In Proceedings of the Thirty Fifth Annual*

Symposium on Foundations of Computer Science, pages 75–85, 1994.

[BFR$^+$02] Michael A. Bender, Antonio Fernández, Dana Ron, Amit Sahai, and Salil Vadhan. The Power of a Pebble: Exploring and Mapping Directed Graphs. *Information and Computation*, 176(1):1 – 21, 2002.

[CGG$^+$15] Jurek Czyzowicz, Leszek Gasieniec, Konstantinos Georgiou, Evangelos Kranakis, and Fraser MacQuarrie. The beachcombers' problem: Walking and searching with mobile robots. *Theoretical Computer Science*, 608, Part 3:201 – 218, 2015. Structural Information and Communication Complexity.

[Che04] Yann Chevaleyre. Theoretical analysis of the multi-agent patrolling problem. In *Intelligent Agent Technology, 2004.(IAT 2004). Proceedings. IEEE/WIC/ACM International Conference on*, pages 302–308. IEEE, 2004.

[Chv75] V Chvátal. A combinatorial theorem in plane geometry. *Journal of Combinatorial Theory, Series B*, 18(1):39 – 41, 1975.

[CLP11] Jurek Czyzowicz, Arnaud Labourel, and Andrzej Pelc. Optimality and competitiveness of exploring polygons by mobile robots. *Information and Computation*, 209(1):74–88, jan 2011.

[DDK$^+$13] Dariusz Dereniowski, Yann Disser, Adrian Kosowski, Dominik Pajak, and Przemyslaw Uznanski. Fast Collaborative Graph Exploration. In *Automata, Languages, and Programming*, volume 7966 of *Lecture Notes in Computer Science*, pages 520–532. Springer Berlin Heidelberg, 2013.

[DDK15] Shantanu Das, Dariusz Dereniowski, and Christina Karousatou. Collaborative exploration by energy-constrained mobile robots. In Christian Scheideler, editor, *Structural Information and Communication Complexity*, volume 9439 of *Lecture Notes in Computer Science*, pages 357–369. Springer International Publishing, 2015.

[Dij59] Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[DKHS06] M. Dynia, J. Kutylowski, F.Meyerauf Heide, and C. Schindelhauer. Smart Robot Teams Exploring Sparse Trees. In *Mathematical Foundations of Computer Science 2006*, volume 4162 of *Lecture Notes in Computer Science*, pages 327–338. Springer Berlin Heidelberg, 2006.

[DKK06] Christian a. Duncan, Stephen G. Kobourov, and V. S. Anil Kumar. Optimal constrained graph exploration. *ACM Transactions on Algorithms*, 2(3):380–402, jul 2006.

[DKP91] X. Deng, T. Kameda, and C. Papadimitriou. How to learn an unknown environment. In *Foundations of Computer Science, 1991. Proceedings., 32nd Annual Symposium on*, pages 298 –303, October 1991.

[DKS06] Miroslaw Dynia, Miroslaw Korzeniowski, and Christian Schindelhauer. Power-aware collective tree exploration. In Werner Grass, Bernhard Sick, and Klaus Waldschmidt, editors, *Architecture of Computing Systems - ARCS 2006*, volume 3894 of *Lecture Notes in Computer Science*, pages 341–351. Springer Berlin Heidelberg, 2006.

[DLS07] Miroslaw Dynia, Jakub Lopuszanski, and Christian Schindelhauer. Why robots need maps. In *Proceedings of the 14th international conference on Structural information and communication complexity*, SIROCCO'07, pages 41–50, Berlin, Heidelberg, 2007. Springer-Verlag.

[DP90] X. Deng and C.H. Papadimitriou. Exploring an unknown graph. In *Foundations of Computer Science, 1990. Proceedings., 31st Annual Symposium on*, pages 355 –361 vol. 1, oct 1990.

[DP04] Anders Dessmark and Andrzej Pelc. Optimal graph exploration without good maps. *Theor. Comput. Sci.*, 326:343–362, October 2004.

[EAK09] Yehuda Elmaliach, Noa Agmon, and Gal A Kaminka. Multi-robot area patrol under frequency constraints. *Annals of Mathematics and Artificial Intelligence*, 57(3-4):293–320, 2009.

[exp16] exploration. *Merriam-Webster.com.* Merriam-Webster, 5 April 2016.

[FGKP06] Pierre Fraigniaud, Leszek Gąsieniec, Dariusz R. Kowalski, and Andrzej Pelc. Collective tree exploration. *Netw.*, 48:166–177, October 2006.

[FHK76] G.N. Frederickson, Matthew S. Hecht, and Chul E. Kim. Approximation algorithms for some routing problems. In *Foundations of Computer Science, 1976., 17th Annual Symposium on*, pages 216–227, Oct 1976.

[FIP08] P Fraigniaud, D Ilcinkas, and a Pelc. Tree exploration with advice? *Information and Computation*, 206(11):1276–1287, nov 2008.

[FRT03] Jittat Fakcharoenphol, Satish Rao, and Kunal Talwar. A tight bound on approximating arbitrary metrics by tree metrics. In *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, pages 448–455. ACM, 2003.

[FT05] Rudolf Fleischer and Gerhard Trippen. Exploring an Unknown Graph Efficiently. In Gerth Brodal and Stefano Leonardi, editors, *Algorithms – ESA 2005*, volume 3669 of *Lecture Notes in Computer Science*, pages 11–22. Springer Berlin / Heidelberg, 2005. 10.1007/11561071_4.

[FW12] Klaus-Tycho Förster and Roger Wattenhofer. Directed graph exploration. In Roberto Baldoni, Paola Flocchini, and Ravindran Binoy, editors, *Principles of Distributed Systems*, volume 7702 of *Lecture Notes in Computer Science*, pages 151–165. Springer Berlin Heidelberg, 2012.

[GR03] Yoav Gabriely and Elon Rimon. Competitive on-line coverage of grid environments by a mobile robot. *Comput. Geom. Theory Appl.*, 24(3):197–224, April 2003.

[Guo95] Yang GuoXing. Transformation of multidepot multisalesmen problem to the standard travelling salesman problem. *European Journal of Opera-*

tional Research, 81(3):557 – 560, 1995.

[HKLT12] Yuya Higashikawa, Naoki Katoh, Stefan Langerman, and Shin-ichi Tanigawa. Online graph exploration algorithms for cycles and trees by multiple searchers. *Journal of Combinatorial Optimization*, pages 1–16, 2012.

[Hoe63a] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58:13–30, 1963.

[Hoe63b] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.

[Kar72] Richard M Karp. *Reducibility among combinatorial problems.* Springer, 1972.

[KKMZ10] Agnieszka Kolenderska, Adrian Kosowski, Michal Malafiejski, and Pawel Zylinski. An improved strategy for exploring a grid polygon. In Shay Kutten and Janez Žerovnik, editors, *Structural Information and Communication Complexity*, volume 5869 of *Lecture Notes in Computer Science*, pages 222–236. Springer Berlin Heidelberg, 2010.

[Lon98] Sven Loncaric. A survey of shape analysis techniques. *Pattern Recognition*, 31:983–1001, 1998.

[Lov96] L. Lovász. Random Walks on Graphs: A Survey. In D. Miklós, V. T. Sós, and T. Szőnyi, editors, *Combinatorics, Paul Erdős is Eighty*, volume 2, pages 353–398. János Bolyai Mathematical Society, Budapest, 1996.

[MRZD03] Aydano Machado, Geber Ramalho, Jean-Daniel Zucker, and Alexis Drogoul. Multi-agent patrolling: An empirical analysis of alternative architectures. In Jaime Simão Sichman, Françcois Bousquet, and Paul Davidsson, editors, *Multi-Agent-Based Simulation II*, volume 2581 of *Lecture Notes in Computer Science*, pages 155–170. Springer Berlin Heidelberg, 2003.

[MU05] Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis.* Cambridge University Press, New York, NY, USA, 2005.

[New60] Donald J Newman. The double dixie cup problem. *American Mathematical Monthly*, pages 58–61, 1960.

[OS12] Christian Ortolf and Christian Schindelhauer. Online Multi-robot Exploration of Grid Graphs with Rectangular Obstacles. In *Proceedings of the Twenty-fourth Annual ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '12, pages 27–36, New York, NY, USA, 2012. ACM.

[OS14] Christian Ortolf and Christian Schindelhauer. A recursive approach to multi-robot exploration of trees. In MagnúsM. Halldórsson, editor, *Structural Information and Communication Complexity*, volume 8576 of *Lecture Notes in Computer Science*, pages 343–354. Springer International Publishing, 2014.

[OS15] Christian Ortolf and Christian Schindelhauer. Strategies for parallel unaware cleaners. In Jie Gao, Alon Efrat, Sándor P. Fekete, and Yanyong Zhang, editors, *Algorithms for Sensor Systems*, volume 8847 of *Lecture Notes in Computer Science*, pages 39–56. Springer Berlin Heidelberg, 2015.

[PR11a] D. Portugal and R.P. Rocha. On the performance and scalability of multi-robot patrolling algorithms. In *Safety, Security, and Rescue Robotics (SSRR), 2011 IEEE International Symposium on*, pages 50–55, Nov 2011.

[PR11b] David Portugal and Rui Rocha. A survey on multi-robot patrolling algorithms. In LuisM. Camarinha-Matos, editor, *Technological Innovation for Sustainability*, volume 349 of *IFIP Advances in Information and Communication Technology*, pages 139–146. Springer Berlin Heidelberg, 2011.

[Pri57] Robert Clay Prim. Shortest connection networks and some generalizations. *Bell system technical journal*, 36(6):1389–1401, 1957.

[PY91] Christos H. Papadimitriou and Mihalis Yannakakis. Shortest paths without a map. *Theor. Comput. Sci.*, 84:127–150, July 1991.

[res16] research. *Merriam-Webster.com.* Merriam-Webster, 5 April 2016.

[RKSI93] Nageswara S. V. Rao, Srikumar Kareti, Weimin Shi, and S. Sitharama Iyengar. Robot Navigation in Unknown Terrains: Introductory Survey of Non-Heuristic Algorithms. Technical Report ORNL/TM-12410:1–58, Oak Ridge National Laboratory, July 1993.

[The15] Felix Thein. A Simulation Environment for the Parallel Unaware Cleaner Problem, June 2015. http://archive.cone.informatik.uni-freiburg.de/pubs/theses/2015-MA-Felix.Thein-A.Simulation.Environment.for.the.Parallel.Unaware.Cleaner.Problem.pdf.

[Yao77] Andrew Chi-Chin Yao. Probabilistic computations: Toward a unified measure of complexity. In *Foundations of Computer Science, 1977., 18th Annual Symposium on*, pages 222 –227, 31 1977-nov. 2 1977.

Appendix

Table 4.1 extended

We restricted the model to small polynomial values for d and k. If this restriction is lifted we can extend Table 4.1 some more for sub-polynomial and super-poly-logarithmic values for the Yo* Algorithm:

Line 1 of Table 4.2

If d and k are not polynomial in n The competitive factors have to be expressed differently.

$$\begin{aligned} d = &\quad k^{\mathcal{O}(1)} \\ k = &\quad d^{\Omega(1)} \\ \Rightarrow \text{competitive factor of Yo*} = &\quad k^{o(1)} \log n \end{aligned}$$

Theorem 6 states a competitive bound of

$$2^{(2+o(1))\sqrt{(\log d)(\log\log k)}}(\log k)(\log k + \log n) \ .$$

for the Yo* algorithm. Note that the requirements for d and k imply: $\log d = \mathcal{O}(\log k)$, which implies $\sqrt{(\log d)(\log\log k)} = \mathcal{O}(\sqrt{(\log k)(\log\log k)}) \in o(\log k)$. Hence, we have a competitive factor of at most

$$\begin{aligned} 2^{(2+o(1))o(\log k)}(\log k)(\log k + \log n) \leq &\quad k^{o(1)}(\log k)^2 \log n \\ \leq &\quad k^{o(1)+\frac{2\log\log k}{\log k}} \log n \\ = &\quad k^{o(1)} \log n \ . \end{aligned}$$

k	d	n	competitive factor	
$d^{\Omega(1)}$	$k^{\mathcal{O}(1)}$		$k^{o(1)} \log n$	Yo*
	$2^{\frac{c^2}{4}(\log k)^2/\log\log k}$		$k^{c(1+o(1))} \log n$	Yo*
$2^{\omega\left(\sqrt{\log d \log\log d}\right)}$		$2^{\mathcal{O}(2^{\sqrt{\log d}})}$	$k^{o(1)}$	Yo*

Table 4.2: Competitive exploration time ratios for the Yo* algorithm

Line 2 of Table 4.2

For very high trees the Yo* shows a polynomial competitiveness. Substituting the result of Theorem 6

$$2^{(2+o(1))\sqrt{(\log d)(\log\log k)}}(\log k)(\log k + \log n)$$

with value:

$$\begin{aligned} d = & \quad 2^{\frac{c^2}{4}(\log k)^2/\log\log k} \\ \Rightarrow \log d = & \quad \frac{c^2}{4}(\log k)^2/\log\log k \\ \Rightarrow \text{competitive factor of Yo*} = & \quad k^{c(1+o(1))}\log n \end{aligned}$$

For the first factor we get the following

$$\begin{aligned} 2^{(2+o(1))\sqrt{(\log d)(\log\log k)}} = & \quad 2^{(2+o(1))\sqrt{\frac{c^2}{4}(\log k)^2}} \\ = & \quad 2^{c(\log k)(1+o(1))} \end{aligned}$$

The other two factors can be bound as following

$$\begin{aligned} (\log k)(\log k + \log n) \leq & \quad (\log k) 2 \log n \\ \leq & \quad k^{\frac{\log\log k}{\log k}} 2 \log n \\ \in & \quad k^{o(1)} \log n \end{aligned}$$

This results in an overall competitive ratio of $k^{c(1+o(1))}\log n$.

Line 3 of Table 4.2

For trees with small depths compared to k Theorem 6 implies the same bound as in the polynomial case.

$$\begin{aligned} k = & \quad 2^{\omega\left(\sqrt{(\log d)(\log\log d)}\right)} \\ \Rightarrow \log k = & \quad \omega\left(\sqrt{(\log d)(\log\log d)}\right) \\ n = & \quad 2^{\mathcal{O}(2^{\sqrt{\log d}})} \\ \Rightarrow \log n = & \quad \mathcal{O}(2^{\sqrt{\log d}}) \\ \Rightarrow \text{competitive factor of Yo* } = & \quad k^{o(1)} \end{aligned}$$

This k implies

$$(\log d)(\log\log d) = o((\log k)^2) \ .$$

If $\log\log d = o(\log\log k)$, then $\log d = o(\log k)$ and therefore

$$\log d = o\left(\frac{(\log k)^2}{\log\log k}\right) \ .$$

Otherwise, if $\log\log d = \Omega(\log\log k)$, then $\frac{1}{\log\log d} = \mathcal{O}(\frac{1}{\log\log k})$ and therefore also

$$\log d = \frac{o(\log k)^2}{\log\log d} = o\left(\frac{(\log k)^2}{\log\log k}\right)$$

follows. We substitute this into the first factor of Theorem 6 and get:

$$2^{(2+o(1))\sqrt{(\log d)(\log\log k)}} = 2^{(2+o(1))o(\log k)} = k^{o(1)} \ .$$

Combined with $\log d = o\left((\log k)^2/(\log\log k)\right)$ gives

$$\begin{aligned} \log n = & \quad \mathcal{O}(2^{\sqrt{\log d}}) \\ = & \quad \mathcal{O}(2^{\sqrt{o((\log k)^2/(\log\log k))}}) \\ = & \quad 2^{o(\log k)} \\ = & \quad k^{o(1)} \ . \end{aligned}$$

Of course, $\log k = k^{o(1)}$ which implies the competitive ratio of $k^{o(1)}$ for the product of these terms.