GDAŃSK UNIVERSITY OF TECHNOLOGY

The author of the doctoral dissertation: Robert Ostrowski
Scientific discipline: Informatics

DOCTORAL DISSERTATION

Title of doctoral dissertation: Multi-agent strategies for selected network problems

Title of doctoral dissertation (in Polish): Wieloagentowe strategie dla wybranych problemów sieciowych

Supervisor *signature*	Second supervisor *signature*
prof. dr hab. inż. Dariusz Dereniowski	
Auxiliary supervisor *signature*	Cosupervisor *signature*

Gdańsk, year 2024

Multi-agent strategies for selected network problems

Doctoral dissertation

author: Robert Ostrowski
supervisor: prof. dr hab. inż. Dariusz Dereniowski

Gdańsk University of Technology
Faculty of Electronics, Telecommunications and Informatics
Department of Algorithms and Systems Modelling

Gdańsk, Poland
2024

Acknowledgements

The dedication of this work is twofold. First, I wish to distinguish those who kept hope when mine has faltered. Especially my supervisor, Dariusz Dereniowski.

Second, I wish to commemorate Jurek Czyzowicz, whom I knew too briefly.

Abstrakt

Przed grupą mobilnych jednostek, zwanych *agentami*, postawiony jest problem wymagający podróży po sieci modelowanej przez graf. Czy istnieje sekwencja ruchów, czy też *strategia*, która rozwiązuje taki problem? Otrzymaliśmy wyniki dla dwóch specyficznych problemów z szerszych obszarów badań do których pasuje powyższy, bardzo ogólny, opis. Poza nimi w pracy znajduje się także przegląd literatury dotyczącej *przeszukiwania grafów* oraz *komunikacji z ograniczoną energią* poprzez przemieszczanie się po grafie. Pomimo tego że autorskie badania zajmują się tylko problemami scentralizowanymi, przegląd obejmuje też rozproszone aspekty odpowiadających im dziedzin badań.

Pierwszym rozważanym problemem jest *heterogeniczne przeszukiwanie grafów*, w którym agenty mają za zadanie odnaleźć uciekiniera w grafie, w którym krawędzie mogą przebyć jedynie agenty określonego typu. To ograniczenie modelowane jest poprzez uzupełnienie klasycznego modelu przeszukiwania krawędziowego o nadanie etykiet krawędziom. Co więcej, algorytm uzyskuje możliwość przypisania agentom etykiet przed wykonaniem strategii. Agent może przebyć, a tym samym wyczyścić, tylko krawędź opatrzoną taką samą etykietą jak jego własna. Ponieważ uciekinier nie jest w żaden sposób ograniczony, jest to uogólnienie klasycznego problemu, a w konsekwencji problem niemniej trudny. Okazuje się, że granica pomiędzy obliczeniowo łatwymi i trudnymi przypadkami przebiega w klasie drzew, stając się interesującym polem badań. Przytaczamy przykład drzewa z trzema etykietami pokazujący, że problem jest niemonotoniczny. Co więcej, liczba ponownych skażeń krawędzi jest rzędu $O(n^2)$. W przeciwieństwie do klasycznego problemu, heterogeniczne przeszukiwanie drzew jest NP-Trudne. Ponadto, pozostaje NP-Zupełne nawet gdy tylko monotoniczne strategie są brane pod uwagę. Te trudności znikają gdy wszystkie krawędzie z jednakowymi etykietami tworzą spójne poddrzewo. W takim przypadku istnieje algorytm wielomianowy, także dla wersji problemu, w której przeszukany podgraf musi być spójny.

W drugim problemie agenty mają wykonać *plotkowanie*, czyli rozpropagować informacje z każdego węzła sieci do wszystkich pozostałych (protokół komunikacyjny "każdy z każdym"). Sieć jest drzewem z wagami na krawędziach i dane mogą być przekazane tylko poprzez podróżujące agenty. Agent zbiera informacje z odwiedzonych węzłów i zostawia je na każdym węźle, na którym w przyszłości się znajdzie. Aby przebyć krawędź agent musi zużyć energię proporcjonalną do jej długości. Energia ta pochodzi z *baterii* danego agenta. Dodatkowo, agenty mogą

przekazywać sobie energię przy spotkaniu. Początkowa konfiguracja agentów obejmuje ich położenia oraz zasoby energii (możliwie różne) przypisane każdemu z nich. W tym ostatnim sensie, problem jest także heterogeniczny. W naszym algorytmie istnieje jeden wierzchołek w którym wszystkie dane zostają zgromadzone po raz pierwszy. Większa część pracy poświęcona jest dowodowi, że optymalna strategia plotkowania składa się z, w pewnym sensie, optymalnej strategii *convergcastu* (komunikacja "jeden z każdym") oraz *broadcastu* (komunikacja "każdy z jednym") z tego wierzchołka. Optymalność wspomnianej wyżej strategii convergcastu zapewnić musi minimalne zużycie energii i odpowiednie ustawienie agentów. Nasze wyniki pokazują, że k agentów w n wierzchołkowej sieci może rozwiązać problem w czasie $O(k^2n^2)$. Zaprojektowanie takiego algorytmu było postawione jako pytanie otwarte w [68].

Abstract

A group of mobile entities, or *agents*, is presented with a problem that requires traversing a network, modeled by a graph, to be solved. Is there a sequence of moves, called a *strategy*, which solves the problem? We obtain results for two specific problems in broader fields of research fitting the aforementioned generic description. Besides them, we include a review of the literature concerning *graph searching* and *energy-constrained communication* by means of traversal of graphs. Although the original research is concerned only with centralised problems, the review covers the distributed parts of their respective fields as well.

First, in the *heterogeneous graph searching* problem, the agents, also called searchers, are asked to find a fugitive in a graph with edges accessible only to specific types of agents. In order to model this restriction, the rules of the edge searching problem are augmented by providing an edge-labeling of the graph and an ability to assign labels to searchers by an algorithm prior to the execution of a strategy. A searcher can only slide through, and therefore clean, an edge with a matching label. Since the fugitive is not handicapped, this problem is a strict generalisation of the classical problem and, as such, it is no less hard. Since it is shown that the boundary between computationally easy and hard problems is in the class of trees, we deem it an interesting object of studies. We provide an example with 3 distinct labels which shows the problem to be *non-monotone*. Furthermore, the number of recontamination events can be of the order $O(n^2)$. In contrast to the results for the classical edge searching of trees, the heterogeneous tree searching problem is proved to be NP-Hard. Moreover, it remains NP-Complete even when restricted to monotone strategies. These difficulties disappear when for each label the edges associated with it form a connected subtree. This case admits a polynomial time algorithm, even if a requirement that the searched subgraph needs to be connected is added.

In the second problem the agents are asked to complete *gossiping*, an all-to-all communication, in a tree network, i.e. disseminate information from each node to every node of the network. The network is an edge-weighted tree and the data can only spread by being carried by agents themselves. An agent collects information from every node it visits and deposits in on every node visited afterwards. In order to traverse an edge, an agent needs to spend energy proportional to the edge's weight. Energy is expended from the agent's *battery*, and agents can exchange their supplies with other agents whenever they meet. The initial configuration of

agents is given as their placement and the, possibly different, amount of energy assigned to each of them. In this last sense, the problem is also heterogeneous. In our algorithm, there exists a unique node that receives the complete set of data first. The bulk of the work consists of a proof that an optimal gossiping strategy can be composed of an optimal *convergcast* (all-to-one) strategy followed by a *broadcast* (one-to-all) strategy from this node. The aforementioned convergcast strategy needs to be optimal in regards to the minimum energy used and it is shown that it provides an appropriate placement of the agents. We show that k agents in a network of n nodes can solve this problem in $O(k^2n^2)$ time. Let us note that designing such an algorithm was the subject of an open question in [68].

Summary

The thesis contains six chapters. We begin with a short introduction, where the outline of the thesis is drawn. Chapter 1 surveys centralized graph searching problems, beginning with the classical formulation by Parsons (Section 1.1) and basic models derived from thereof (Section 1.2.2). This latter section also contains an explanation of the concept of monotonicity and a, somewhat outmoded, link of searching to graph pebbling. Section 1.3 explores the connection of search numbers with other graph parameters, pathwidth and treewidth being the most prominent, and graph minor theory. Next, in Section 1.4 we consider the optimization of the time of searching instead of the number of agents. Further, Section 1.5 is concerned with non-monotone searching problems, connected searching being the prime example. Some searching problems that did not fit in the aforementioned sections are relegated to Section 1.6.

Chapter 2 consists of two main parts: a preliminary survey of general, agent adjacent concepts, followed by their further use, exemplified in the field of graph searching. It introduces agents and concepts associated with distributed problems in Section 2.1. We make use of them in Section 2.2 where further, distributed graph searching models are surveyed. This section is subdivided further to reflect collections of works we deem similar. We note that a thread from the last group of self-stabilizing models connects back to Section 1.6.

Chapter 3 opens the survey of energy constrained communication problems with some background and notation used to talk about them in detail. Section 3.1 introduces the specific problems: data delivery, convergcast, broadcast and gossiping. An overview of related works is relegated to their respective sections, namely Sections 3.1.1, 3.1.2, 3.1.3 and 3.1.4. Section 3.2 serves as a connection between the two problems specific to this work by means of providing a very general classification of problems and a brief survey of relevant examples. We note that this chapter discusses some problems rather loosely related to the results of this thesis. However, we included them having as a goal to give a more complete state of the art regarding the concept of heterogeneity.

Chapter 4 opens the presentation of one of the two subjects of the original research, namely *heterogeneous graph searching*. The formal introduction and definitions can be found in Section 4.2.1. Our results are divided into four sections. First, Section 4.3 provides an example of a graph where allowing recontamination reduces the number of searchers needed to successfully clean the graph. The

number of recontamination events occurring in our example is shown to be $O(n^2)$. Next two sections are concerned with reductions. We use the well known 3-SAT problem to prove that monotone and non-monotone variants of the problem for the class of trees are NP-Complete and NP-Hard, respectively. Lastly, Section 4.6 provides a polynomial-time algorithm that finds an optimal heterogeneous search strategy in a given special case, i.e. when each label induces a connected subtree. Furthermore, this approach is effective even if a search strategy has to be connected. The chapter ends with a summary and discussion of open problems.

Chapter 5 is concerned with a specific variant of the gossiping problem outlined in Section 3.1.4. We are interested in gossiping when energy-constrained agents are initially scattered across a weighted tree network. Edge weights dictate how much energy an agent spends to traverse an edge. The initial battery sizes vary but agents can exchange energy whenever they meet. The problem is formalized in Section 5.2 and further necessary notation is given. Next, Section 5.3 summarizes our approach and results. We use the fact that in our problem a gossiping strategy can be divided into two stages: the first corresponding to a convergcast strategy, then followed by the second, based on a broadcast strategy.

Section 5.4 contains the proof of correctness of our analysis. Section 5.4.1 provides a convergcast algorithm used in the first stage of gossiping. Next, Section 5.4.2 is concerned with a broadcast algorithm we use. Let us note that the algorithm itself is not novel and appeared in [68]. Hence, we use only its selected properties. Then, Section 5.4.4 is devoted to proving that joining these two strategies results in an optimal gossiping strategy. In order to do so, we use some of the previously outlined properties of the aforementioned broadcast algorithm. Just as in the previous chapter, summary and open problems close this chapter as well.

Chapter 6 concludes the thesis. The first of its three components is a short list that summarizes the results contained in the two previous chapters. It is then followed by an aggregate of potential applications of the discussed concepts and our work (Section 6.1). Before we proceed, introduction of the third component requires a few sentences of context. The main connections of this thesis to earlier works are as follows. The results contained in Chapter 4 and Chapter 5 have been published — the former in [66] and the latter in [90]. Moreover, the results that eventually appeared as [90] were also presented, by the author of this thesis, at the 11th International Conference on Algorithms and Complexity (CIAC 2019) [19]. Some passages in Chapter 1 originate from [196], the author's early, and much smaller in scope, survey. Last but not least, this thesis includes (in the chapter based on [66]) an answer to the open question of providing a gossiping algorithm stated in [68]. Moreover, we note that the author was a new addition to the research team. Contributions of the author to the above works are outlined in Section 6.2.

The thesis of this dissertation is as follows. A generalization of the graph searching problem where an agent's access to specific subgraphs is restricted is computationally difficult and non-monotone in the case of trees. The polynomial complexity is preserved only in particular cases in the class of trees. Gossiping by energy constrained agents in tree networks can be solved using joint strategies for two subproblems: optimal (in a specific sense) convergcast and broadcast.

Contents

List of Symbols

$V(G)$	set of vertices, or nodes, of graph G. Simply V when the graph is unambiguous
$E(G)$	set of edges of graph G. Simply E when the graph is unambiguous
$e = \{u, v\}$	$e \in E, u \in V, v \in V$. Edge e between two vertices u and v
$G = (V, E)$	(undirected) graph G, an ordered pair of two sets: vertices V and edges E
$n = \|V\|$	number of vertices
$m = \|E\|$	number of edges
k	number of agents
$\mathrm{es}(G)$	edge search number of graph G
$\mathrm{ns}(G)$	node search number of graph G
$\mathrm{ms}(G)$	mixed search number of graph G
$\mathrm{ms}(G)$	mixed search number of graph G
$\mathrm{sms}(G)$	strong mixed search number of graph G
$\mathrm{vns}(G)$	visible search number of graph G
$\mathrm{avms}(G)$	mixed search number against an agile and visible fugitive of graph G
$\mathrm{ies}(G)$	internal search number of graph G
$\mathrm{mes}(G)$	monotone edge search number of graph G
$\mathrm{mies}(G)$	monotone internal edge search number of graph G
$\mathrm{cs}(G)$	connected edge searching of graph G
$\mathrm{ices}(G)$	internal connected edge search number of graph G
$\mathrm{mces}(G)$	monotone connected edge search number of graph G
$\mathrm{mices}(G)$	monotone internal connected edge search number of graph G
$\mathrm{fsn}(G)$	fast search number of graph G
$\mathrm{fesn}(G)$	fast edge search number of graph G
$\mathrm{fms}(G)$	fast-mixed search number of graph G
$\mathrm{fet}(G)$	fast edge search time of graph G
$\mathrm{mpbw}(G)$	minimum progressive black and white pebble demand of graph G

$\mathrm{mpb}(G)$	minimum progressive black and the white pebble demand of graph G
$\mathrm{tb}(G)$	topological bandwidth of graph G
$\mathrm{pw}(G)$	pathwidth of graph G
$\mathrm{tw}(G)$	treewidth of graph G
$\mathrm{vsep}(G)$	vertex separator of graph G
G	search cost of graph G
$\mathrm{esc}(G)$	minimum edge search cost of graph G
$\mathrm{mit}_G(k)$	monotone immunity number of graph G for decontamination with k agents
$\mathrm{it}_G(k)$	immunity number of graph G for decontamination with k agents
B_i	battery size of i-th agent
B	battery size of all agents when it is uniform
Ω, Θ, O, o	asymptotic notation symbols

Symbols in Chapter 4

z	number of colors — different labels that can be assigned to agents and edges
$c\colon E(G) \to \{1, \ldots, z\}$	function that assigns colors to the edges of G
$G = (V(G), E(G), c)$	edge-labeled graph G
$c(e), e \in E$	color of an edge e
$c(v), v \in V$	set of colors of a node (vertex) v
$\tilde{c}(s)$	color of a searcher s
$\mathcal{S} = (m_1, \ldots, m_\ell)$	search strategy
m_i	i-th move in a strategy, also move i
$\mathcal{D}_i$	set of contaminated edges after move m_i
$\mathcal{C}_i$	set of clean edges after move m_i
$\mathcal{R}_i$	set of recontaminated edges after move m_i
$\mathtt{hs}(G)$	heterogeneous search number of graph G
$\mathtt{hcs}(G)$	heterogeneous connected search number of graph G
$\mathtt{mhs}(G)$	monotone heterogeneous search number of graph G
$\mathtt{mhcs}(G)$	monotone heterogeneous connected search number of graph G
$\beta(G)$	lower bound on a heterogeneous search number number of G

Symbols in Sections 4.4 and 1.5

T'	subtree of tree T
3-SAT	Boolean Satisfiability Problem where each clause is limited to at most three literals
n	number of variables in an instance of 3-SAT
m	number of clauses in an instance of 3-SAT
x_p	$p \in \{1, \ldots, n\}$, p-th variable p in an instance of 3-SAT
$\overline{x_p}$	negation of x_p
$l_{i,j}$	j-th literal in the i-th clause in an instance of 3-SAT, either x_p or $\overline{x_p}$

l	literals
$C_i = (l_{i,1} \vee l_{i,2} \vee l_{i,3})$	$i \in \{1, \ldots, m\}$, i-th clause in an instance of 3-SAT
T_{SAT}, $\tilde{T}_{\text{SAT}}$	our constructed trees corresponding to an instance of 3-SAT
$\mathcal{Q}$	set of colors
$\mathtt{V}_p$	color corresponding to x_p
$\mathtt{T}_p$	color corresponding to x_p being true
$\mathtt{F}_p$	color corresponding to x_p being false
$\mathtt{C}_i$	color corresponding C_i
$\mathtt{R}, \mathtt{O}_{i,j}$	$i \in \{1, \ldots, n\}$, $j \in \{1, 2\}$ remaining colors
Symbols in Chapter 5	
T	tree
r	root of T
$v \prec u$	$v, u \in V(T)$, v is a *descendant* of u in T — u lies on the path between v and r
$\text{par}(v) = u$	$v, u \in V(T)$, $v \prec u$ in T and u, v are adjacent
T_v	$v \in V(T)$, the subtree of T induced by v and all its descendants
$w(e)$	$e \in E(T)$, weight of e
$\mathcal{C}_{\text{alg}}$, $\mathcal{C}_{\text{min}}$, $\mathcal{C}_{\text{opt}}$	some convergcast strategies
$\mathcal{B}_{\text{min}}$, $\mathcal{B}_{\text{opt}}$, $\mathcal{B}_{\text{alg}}$	some broadcast strategies
$\mathcal{G}_{\text{opt}}$, $\mathcal{G}_{\text{alg}}$	some gossiping strategies
$\downarrow_e(\mathcal{S})$	$e \in E(T)$, the number of downward traversals of e in a strategy $\mathcal{S}$
$\uparrow_e(\mathcal{S})$	$e \in E(T)$, the number of upward traversals of e in a strategy $\mathcal{S}$
$\text{energy}(a)$	energy of an agent a
$\text{end-energy}(\mathcal{S})$	sum of energy levels of all agents at the end of a strategy $\mathcal{S}$
$\text{cost}(\mathcal{S})$	*cost* of $\mathcal{S}$

Introduction

First and foremost, the subject of this thesis is deeply rooted in graph theory, since we use its language to describe the titular networks. We assume the readers familiarity with the standard graph-theoretic notation and concepts that can be found in [229] which we employ throughout this work. Likewise, we use the classical notions regarding discrete mathematics, computational complexity, algorithmic analysis from [62, 198].

The obtained results concern two specific graph problems, *heterogeneous graph searching* and *gossiping with energy-constraints*, which are posed to a team of entities. Jointly, we will refer to these entities as mobile agents, in order to put emphasis on their properties that can make them non-interchangeable and their ability to change their position in a graph. Each of the two problems belongs to a different, broader, field of study: the former is a part of investigation of graph searching and the latter can be construed as a problem of energy-constrained traversal of a graph. The relevant literature is reviewed in Chapters 1, 2 and 3. We note that the review of literature about graph searching includes, relatively minor, passages from the author's early survey [196]. Despite the fact that both of our problems are centralised, the domain of agent problems is closely associated with distributed computing and our review of literature encompasses a handful of such models. Hence, we make an effort to introduce the relevant terms. Nonetheless, for standard terms that we have skipped we refer the reader to [210].

The mobility of the agents lends itself to the description of the solution in terms of a sequence of moves, called a *strategy*. We note, however, that our problems share little notation beyond basic definitions. Hence, we deffer the discussion of other relevant concepts and particular terms to each problem's respective chapters. A faint thread connecting the presented original research problems of this thesis is a high level idea of heterogeneity among entities used to solve the problems. This concept is elucidated in Section 3.2, which contains an overview of a miscellaneous collection of problems from disparate fields that fit such a classification.

In the interest of not overloading the reader with unfamiliar terms before any background is introduced let us stop this chapter short of a preliminary discussion of our specific models. Nonetheless, we trust that readers well versed in the relevant fields of research may safely proceed to either Chapter 4 or 5, where the original results are discussed. Further guidance can be found in the Summary preceding the main body of this thesis or Sections 5.1 and 4.1.1. Likewise, each of these two

chapters contains their own concluding section. Thus, Chapter 6 provides only a short list of main results, a brief discussion of potential applications and an outline of contributions.

Chapter 1

Graph searching

1.1 Introduction to graph searching

Roots of graph searching as a mathematical problem can be traced to woes of Richard Breisch [45] — T.D. Parsons' spelunker friend. The problem, raised in 1967, seeks to find the minimal number of rescuers needed to find, with certainty, a man lost in a cave. Searchers are familiar with the structure of cave; however, they have no information about whereabouts or actions of the lost person. This last point led Parsons to assume the worst case scenario, that *"The searchers must proceed according to a predetermined plan which will capture the lost man even if he were an arbitrarily fast, invisible evader who, clairvoyant, knows the searcher's every move [199]."* Thus, the lost man became an evader, a fugitive. Another assumption made by Parsons was that the cave can be represented as a graph on which entities move continuously and capturing the fugitive is realized by being in the same space as him — the only way to capture the fugitive is to bump into him in the darkness.

Given this model, it is not hard to imagine altering the problem to obtain a different version, thus ensuring a plethora of open questions following each publication. Moreover, it is not obvious when given problems are equivalent or how closely are they related. In fact, as described by Fomin et al. in [128], a problem studied independently by N. Petrov in 1982 in [202] was proven to be equivalent to the described model in [141, 140][1], but it is not the only such case we will investigate.

In the case of Parsons' problem, thinking about a fugitive with unbounded speed as an entity with a defined location can be misleading. An alternative story poses a question whether a system of pipes contaminated by a toxic gas, which spreads whenever it is not contained, can be cleaned by traversing through the environment. This formulation was introduced by Parsons himself in [199], and nowadays it often takes precedence over the traditional cave narrative in the

[1] Due to the language barrier we were not able to verify this claim directly.

case of distributed problems, labeled as decontamination. Another example of its prominence can be found in the used terminology. Edges which are not accessible to fugitive are often referred to as clean and those which can be occupied by him as contaminated. Furthermore, when the fugitive can reach a previously clean edge again, it is said to be recontaminated. Whenever no recontamination occurs during search, it is called monotone. These formulations imply an omnipresence of the fugitive; hence, it has no effective agency in its movement. Therefore, graph searching problems can be considered 1-player games.

Another branch of related problems are cop and robber games, where two players alternate moves. Together they can be labeled as pursuit-evasion problems; however, both of these branches differ considerably. As a general rule, all cops and robber games are beyond the scope of this work, and we refer the interested reader to the rich body of literature on associated problems, such as [39].

1.2 Basic models

1.2.1 Edge searching

In Parsons' case the motivation of centering the problem on the number of searchers is explained by a simple example. Given a circular cave one searcher will not find a man who matches his speed and direction. However, two searchers can formulate a plan in which one of them stays in place, preventing the fugitive from passing him, while the other moves around the cave. The same result can be achieved, likely faster, when both searchers move in opposite directions. By the time they meet they are certain to find the fugitive. However, the question of what the plan is or how fast can it be executed is secondary to the question of whether it exists for a given cave and a number of searchers at all. That being said, the other questions were explored and will be summarized later as well.

Parsons [199] started the formal description of the problem by embedding a graph in a three dimensional space, thereby relating the problem to topology, which is beyond the scope of this work. We just note that such an embedding is possible for any graph. Next, he defines a search plan in terms of a family of continuous functions, one function for each searcher, describing its position in a given time. For every possible function describing the movement of an evader, there exists a time when position of one of the searchers and evader are equal. Thus, Parsons is interested in the minimum cardinality of this family of functions for a given graph.

Instead of Parsons' highly formalised description it may be more convenient to use a game-like approach, which will be easier to relate to other models. Note that the original version of the game is meant to be continuous; however, a discreet variant is more popular. Searchers act according to a *strategy*, which is a sequence of *moves*. Each move can be one of the following actions:

1. placing a searcher on a vertex,
2. removing a searcher from a vertex,

3. sliding a searcher present on a vertex along an edge of the graph, which results in a searcher ending up on its other endpoint.

This approach was used by Megiddo et al. in [186], and was related to a description of pebbling problems (see Section 1.2.2). Initially, the fugitive can hide in the whole graph — the graph is *contaminated*. During the game a sliding searcher can traverse an edge, thereby making sure that this edge cannot be occupied by the fugitive. In this case we say that the edge is *clean*. However, if there exists a path from a contaminated to a clean area which is unobstructed by the searchers, then the fugitive can hide there again (we say that the edge became *recontaminated*). If no recontamination occurs, then the search strategy is *monotone*. Note that monotone strategies always require at most a polynomial number of moves, and as such, monotonicity is often a desired property — in fact some works deliberately restrict considered strategies to only monotone ones. Searchers win if they can clean the whole graph, and we are interested in the minimal number of searchers such that there exists a winning strategy. This parameter of a graph was called the search number by Parsons in [199]; however, to further distinguish it from other parameters, it will be called the *edge search number* of graph G, or simply denoted as $\mathrm{es}(G)$, in this work. Let us introduce one more term for the purpose of simplifying informal descriptions. We say that a searcher *guards* a clean subgraph S if their removal would cause recontamination of some edge of S.

The result obtained by Parsons [199] is a characterization of the edge search number for trees. Parsons research had not exhausted the problem he had outlined, and a few further breakthroughs were made in the following years, with major contributions of Megiddo et al. [186][2] and LaPaugh [171]. While it was shown that a strategy that cleans a tree graph T using $\mathrm{es}(T)$ searchers can be constructed in $O(n \log n)$ [186], the preliminary proof that the problem is NP-Complete for arbitrary graphs was initially incomplete. It was not known whether the problem belonged to NP. In fact, some of the aforementioned pebbling problems, despite being described in a language similar to searching problems, turned out not to be in NP, as was shown in [175] and, much later, in [146]. If recontamination is possible, then the number of moves required to execute a non-monotone strategy might not be polynomial in the size of a graph. The conjecture that recontamination does not help to search a graph, which made for a catchy title [171], was proven later and was acknowledged in the final version of the former article, thus closing the question of NP-completeness. In the same work, a characterization of graphs with search number less or equal to 3 was given. Furthermore, it was followed by a linear time algorithm recognizing such graphs. We will refer to problems where allowing recontamination to occur yields no decrease of its respective search number as *monotone*; thus, we can say that the (edge) graph searching problem is monotone.

[2]See Section 1.3.3 for further information on both Parsons' and Megiddo's contribution viewed from a different angle.

1.2.2 Basic monotone variants

A node search game and the associated *node search number*, denoted as $\mathrm{ns}(G)$ for graph G, were introduced in [167]. In this version, only actions of placing and removing a searcher from a vertex are needed. An edge is considered clean when both of its endpoints are occupied by searchers at the same time. Thus, the available moves are simply:

1. placing a searcher on a vertex,
2. removing a searcher from a vertex.

Authors prove that the edge and node searching are closely related. In fact, given a strategy for the former, one can obtain a strategy for the latter and vice versa by introducing at most one additional searcher. From this transformation it follows that the property of monotonicity applies to both node and edge searching.

Another corollary from the possibility of transforming strategies is the fact that, for a given graph G, both search numbers differ by at most one, i.e.

$$\mathrm{ns}(G) - 1 \leq \mathrm{es}(G) \leq \mathrm{ns}(G) + 1.$$

The examples that the bounds are tight are very simple. A singular edge requires one searcher in the edge searching model and two in node searching; however, in the case of a complete bipartite graph on 6 vertices, the relation is reversed and edge searching requires one more searcher. Authors note that, in general, edge searching outperforms the node variant only when a searcher slides through the last contaminated edge adjacent to a vertex. Unsurprisingly, it follows that finding node search number is NP-Complete, just like its edge counterpart.

Leaving pebbles behind

The node search number coincides with parameters in particular pebbling games. While pebbling and searching share a similar language used to describe problems, drawing parallels between them did not bear many results — the following results were presented in a single article [167]. Rather than overloading the work with definitions, we will try to give the reader an intuition why the connection is so tenuous despite the similarities. While [167] contains basic definitions, for an overview of pebbling itself see [156, 216][3]. Both searching and pebbling generally ask for the minimum number of used entities. In the case of pebbles this parameter is called a *pebble demand*. We are interested in specific versions of black pebble demand [138, 212] and black and white pebble demand [61]. In order to find a suitable parameter to equate with the node search number, two problems had to be solved. In general, pebbling games are played on digraphs and can be thought of as models of computation. Since reusing resources during computation can yield benefits, backtracking and pebbling vertices multiple times is expected

[3]Note that following only literature referred to in the original article misses decades of development in the field.

in pebbling problems. In contrast, node searching is monotone, hence the restriction to only consider pebbling strategies, called progressive, that pebble each vertex only once. The second problem is finding a way to compare parameters concerning digraphs and undirected graphs. This issue was solved by taking such a transformation of an undirected graph to a directed one, that the progressive pebble demand was minimal. In conclusion, the obtained result prove that, for a graph G, node search number is equal to the minimum progressive black pebble demand and the minimum progressive black and white pebble demand (denoted $\mathrm{ns}(G) = \mathrm{mpbw}(G) = \mathrm{mpb}(G)$, respectively). Given the apparent necessity of these restrictions placed on pebbling problems, further investigation of the link between searching and pebbling does not seem to be promising to researchers.

A foundation to build upon

A unification of both the node and edge searching models was proposed by Bienstock and Seymour [24]. This work introduced *mixed searching*, where searchers can clean edges by either sliding or occupying both endpoints. The corresponding parameter $\mathrm{ms}(G)$ was called the *mixed search number*. Since in the mixed searching pursuers have capabilities from both edge an node searching, $\mathrm{ms}(G) \leq \mathrm{es}(G)$ and $\mathrm{ms}(G) \leq \mathrm{ns}(G)$. Furthermore, it can be less than them by at most one. The three models described above are based on the premise of the same story, and they are a longstanding inspiration for various extensions and restrictions, thereby creating a rich field of research.

Let us close this section with an introduction of another generalization of mixed searching which appeared in this century. [234] defined strong mixed searching and its parameter $\mathrm{sms}(G)$. In this model, searchers are given one more ability to clean edges, this time of an entire subgraph induced by a vertex v and its neighbours, provided that they occupy every neighbour of v (note that v itself does not have to be visited). The authors establish montonicity and a clear relation to the three aforementioned problems. Furthermore, they propose two further generalizations of the model, one based on visibility, the other on the conditions of capture. To the best of my knowledge these threads were not followed thus far.

Searching Overview		
Model	Complexity	Relations
edge searching, $\mathrm{es}(G)$	NP-C in general [186, 171], polynomial for trees [186]	
node searching, $\mathrm{ns}(G)$	NP-C in general [167], polynomial for trees [211]	$\lvert\mathrm{ns}(G) - \mathrm{es}(G)\rvert \leq 1$
mixed searching, $\mathrm{ms}(G)$	NP-C in general [224], polynomial for trees [224]	$\mathrm{ms}(G) \leq \mathrm{es}(G) - 1$
strong mixed searching, $\mathrm{sms}(G)$	NP-C in general [234] polynomial for trees[234]	$\mathrm{sms}(G) = \mathrm{ns}(G) - 1$

Table 1.1: Basic models of graph searching.

1.3 A wide perspective on graph parameters

1.3.1 Pathwidth and treewidth

A great amount of insight into searching problems was gained using the concept of pathwidth. This parameter was developed in Robertson and Seymour's series of articles on graph minors [206]. While the first article in the main series, regarding pathwidth, does not mention searching explicitly, searching appeared in its followup, on the same subject of excluding a forest, but doing so quickly [22]. There the authors apply their findings to give an alternative proof that node searching is monotone. This is noteworthy, since the original proof [167] was based on the monotonicity of edge searching, proven in [171]. Additionally authors claim that the same technique can be applied to edge searching, and indeed, this result was published later as [24], thereby unifying the proof of montonicity of these two models. A dozen years later, Fomin [127] followed the idea and applied it in an even broader context of games.

This connection with parameters of path and tree decompositions of graphs is one of the main motivations of studying graph searching, thanks to their numerous applications. While a use of graph searching to guide a rescue operation may be dismissed as unlikely, its connection with the aforementioned parameters provides an array of applications, ranging from graph drawing [96], through design of algorithms [23] and electric circuits [108, 81], to a theory in natural language processing [170].

Let us introduce these parameters formally. A path decomposition of a graph G is a sequence of sets $X_1, \ldots, X_r \subseteq V(G)$ such that for $1 \leq i < i' < i'' \leq r$, $X_i \cap X_{i''} \subseteq X_{i'}$ (i. e. each vertex appears only in a continuous subsequence of the whole sequence) and for each edge $\{v, w\} \in E(G)$, some X_i contains both v and w. Finally, each vertex belongs to some X_i. The pathwidth of a graph G, denoted $\mathrm{pw}(G)$, is the minimum integer $k \geq 0$ such that there exists a path decomposition of G which satisfies: $|X_i| \leq k + 1$ for each $i \in \{1, \ldots, r\}$. Thus, pathwidth is the maximum size of a single subset in a decomposition that minimizes this value, additionally decreased by 1 so that the pathwidth of a path graph is intuitively equal to 1.

The definition of tree decomposition is no less complex. For a graph $G = (V, E)$ it is a pair $(\mathcal{Z}, T)$, where $T = (W, F)$ is a tree and $\mathcal{Z}$ is a family of subsets of $V(G)$ such that $Z_1, \ldots, Z_w$, $w \in W$. Note that each index which identifies the subset corresponds to a vertex of T. Furthermore, for each edge $(v, w) \in E(G)$ some Z_i contains both v and w and for each $i, j, k \in W$ if j is on the path from i to k in T , then $Z_i \cap Z_k \subseteq Z_j$. Additionally, each vertex belongs to some Z_i.Analogically to the definition of the pathwidth, the treewidth of a graph G, denoted $\mathrm{tw}(G)$, is the minimum integer $k \geq 0$ such that there exists a tree decomposition of G which satisfies: $|Z_i| \leq k+1$ for each $i \in \{1, \ldots, r\}$. Note that this definition ensures that for any tree T, $\mathrm{tw}(T) = 1$.

Let us address the complexity of pathwidth before proceeding. The result of NP-completeness of pathwidth was obtained in [5]; however, it is also known to be

fixed parameter tractable [36, 35]. In fact, the problem of deciding if $\mathrm{pw}(G) \leq k$ for a given graph G and a fixed integer k can be solved in time linear in terms of n but exponential in k (see Section 1.3.3).

While the definitions of the parameters introduced in this section are not trivial to understand, the intuition behind their connection with searching problems can be explained rather succinctly, using their interpretations as measures of how much the graph in question resembles a path or a tree in the cases of pathwidth and treewidth, respectively. A reader unsatisfied with the brevity of the presented introduction to the topic, but overwhelmed by the sheer volume of the graph minor series, may find [33] an appealing middle ground.

1.3.2 Finding a path

When one seeks to find the fugitive on a path, it is sufficient to place a searcher on one of its ends and move it to the other end. If the path is too wide for a single searcher to block it, a solution is to increase the number of searchers to match its width. Due to different variants of the problem the exact details of this "matching" vary, but almost every graph searching problems can be analysed in a similar fashion to find an equivalent underlying graph parameter. While we admit to posses a certain flair for the dramatic, the usage of almost every here is not a gross exaggeration. Indeed, it is noteworthy to even find a search problem not related to pathwidth in some fashion [183].

The first connection of these parameters for an arbitrary graph G was established using the *vertex separator* (denoted $\mathrm{vsep}(G)$) as an intermediary parameter. The relation between these parameters can be expressed as follows:

$$\mathrm{ns}(G) - 1 = \mathrm{vsep}(G) = \mathrm{pw}(G) = \mathrm{sms}(G).$$

The first result was due to [167], the previously mentioned work investigating connections between pebbling and searching. The second appeared a few years later [164], tightening the bounds provided by [103]. We also note that this last work establishes fixed-parameter tractability of the node searching problem by providing a polynomial time algorithm deciding whether $\mathrm{vsep}(G) \leq k$ for a fixed k. The rightmost part of the equality was established in [234]. If we think of establishing the strong mixed searching as finding a search number matching the pathwidth exactly, then one can also be interested in the reverse — finding a "width" problem matching a given searching model. In this regard, [224] matches mixed searching with the *proper pathwidth parameter*.

1.3.3 Minor interlude

Let us address another contribution in the field obtained from the Graph Minor series. Graph searching appears among related topics in its second article [207], this time on the subject of treewidth. Let us begin with an introduction of an intuitive approach to searching for a fugitive in a tree graph and build gradually. Since we will be referring to trees extensively, let a *branch* be a node with at

least 2 children. Initially we restrict our analysis to trees with degrees of vertices at most three and hope to provide the reader with enough intuition to see that construction of a search strategy in any of the models introduced so far can easily be generalized to work for arbitrary trees.

Since there are no cycles, a single searcher constitutes an insurmountable obstacle which cuts evader's access to some subtree. However, an invisible fugitive can slip away when the searcher enters a wrong subtree when their path branches out, and since the fugitive is omniscient they can ensure that each choice of the single searcher is wrong. A clever reader can thus deduce that the number of required searchers increases as they encounter a vertex connecting more than two subtrees that require the presence of at least every searcher deployed so far.

And indeed, this analysis leads to the idea used by Parsons [199] in his original work in 1978. Based on the fact that a tree has a search number at least $k+1$ if and only if at least three disjoint subtrees connected to one vertex require at least k searchers, he obtained a recursively defined set of trees $\mathcal{T}_i$ for each $i \geq 1$, where each tree in the set has a search number equal to i. In order for a given tree T to have $\mathrm{es}(T) = k$, it has to contain a subtree homeomorphic to a tree in $\mathcal{T}_k$, but no subtree homeomorphic to a tree in $\mathcal{T}_i$ (see Figure 1.1) for $i \geq 1$. Thus, search number is logarithmic in terms of the size of the tree. While Parsons' work was not constructive in terms of providing a practical solution for the proposed problem, a linear time algorithms finding search strategies for both node and edge searching were described in [201].

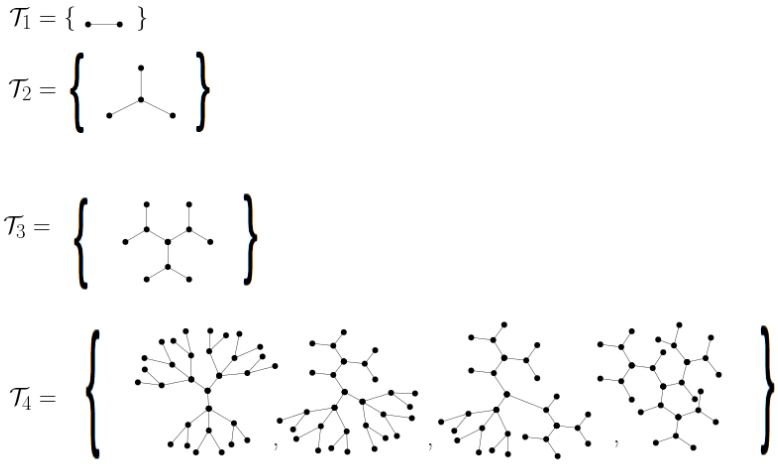


Figure 1.1: First four sets of Parson's family of trees $\mathcal{T}_i$, $i \in \{1, \ldots, 4\}$[199]. Note that the trees in $\mathcal{T}_{i+1}$ consist of connected trees from $\mathcal{T}_i$.

Note that this very early characterization by a family of graphs is consistent with the general idea of forbidden minors developed by Robertson and Seymour and published after 1983. In fact, most graph searching problems are minor closed,

that is: for a given graph G its search number is at most some fixed value k if it doesn't contain any graph from a certain graph family, its *forbidden minors*, characterizing it[4]. An intuitive application of this concept is the property that if a graph H can be obtained by deleting or contracting some edges and removing some isolated vertices of a graph G (i.e. H is a minor of G), then $\mathrm{es}(H) \leq \mathrm{es}(G)$. A more general concept, when the forbidden structures are not necessarily minors, is to define a set of *obstructions*.

Furthermore, the number of forbidden minors is finite, as a consequence of the proof of Wagner's conjecture [208], and thus there exist algorithms which recognize graphs of pathwidth and treewidth equal to k which are polynomial in the size of G but exponential in the size of k. Note that the above results are non-constructive; however, [32, 36] provide explicit polynomial algorithms capable of finding tree and path decompositions of width given by a fixed constant k, if such exist. Parsons notes that, even in the case of trees, determining the size of the set of obstructions "would be a hard combinatorial problem". The optimism of finding ways to efficiently solve minor closed problems in this fashion is tempered by Fellows and Langston in [108], where they show that it is impossible to construct an algorithm finding the set of forbidden minors for a given minor closed family. For more information we refer a curious reader to their further work on this subject [107, 109]. Since many of the theoretical works on exact algorithms have to contend with exponential complexity or huge constants, a more practically minded reader might be interested in a recently developed positive-instance driven dynamic programming approach [225, 9].

In the case of edge searching, there was comparatively little success in finding forbidden minors since Parson's characterization in terms of families of trees in 1978. The complete characterization of graphs G and G' such that $\mathrm{es}(G) \leq 1$, $\mathrm{es}(G') \leq 2$ appeared in the article by Meggido at al. in 1988 [186] (recall that it is the same work that established the early results on the subject of complexity of graph searching, moreover, see Figure 1.2). Additionally, their characterization of bicconnected graphs B (i.e. such that removing any single vertex cannot make the graph disconnected) such that $\mathrm{es}(B) \leq 3$ was expanded by describing biconnected outerplanar graphs B' such that $\mathrm{es}(B') \leq 4$ in 2022 [95]. In contrast, there are more results concerning pathwidth and treewidth [165, 209]. Nonetheless, they are not optimistic, as even in the case of trees the size of the set of forbidden minors for graphs of pathwidth k has a lower bound of $k!^2$ [223].

This line of research is continued in the form of an investigation of properties of graphs that guarantee that their search number is small. To this end one can use the concept of *linear-width* [226, 38, 223], which ties edge, node and mixed searching models. In fact, the algorithm for determining whether for a given graph there exists an edge ordering with linear width of at most some fixed value k [38] can be used to obtain a search strategy in any of these models in linear time. These results are closely related to the fixed parameter traceability of pathwidth, proven in [36] and [32] (note that the latter is also concerned with treewidth).

[4]For a brief summary of the general concept see the following lecture notes [228].

Due to the exponential in k nature of these algorithms, a practical approach to determining linear-width presented in [226] is based on a complete characterization of the obstruction set of graphs with linear-width lesser or equal to 2. This set consist of 57 graphs, where the one with the most vertices is a tree (22 nodes). The authors conjecture that a tree being the biggest obstruction in terms of nodes is the case for any k. A recent description of the state of the art on the topic of obstructions can be found in the PhD thesis of Dimitris Zoros published in 2017 [239], which contains a chapter dedicated to graph searching.

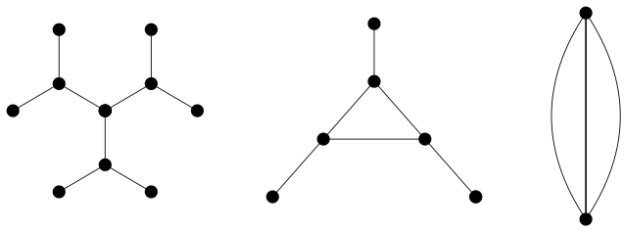


Figure 1.2: The full set of forbidden minors for 2-searchable graphs [186].

1.3.4 I see a path among trees

By now we have hopefully provided the reader with some intuition on optimal strategies of graph searching, despite the fact that the effort needed to prove their correctness is often in contrast with their apparent simplicity. Consider a case when the fugitive is visible in the node search model — this is the premise introduced in [213] and then significantly expanded in [80]. Even two searchers can now clean an arbitrary tree, by checking the position of the evader and choosing the correct branch to pursue them. Generalization of this approach to arbitrary graph G suggests that in order to capture a visible fugitive in G we should find out of how similar to a tree it is, and consequently measure how "wide" this tree is, to find the corresponding *visible search number* (denoted $\mathrm{vns}(G)$). This allows us to expect a result analogous to the relation between the node search number and pathwidth for the case of invisible evader, and indeed, it was formally proved in [213] that $\mathrm{vns}(G) - 1 = \mathrm{tw}(G)$.

Let us call a fugitive able to move only whenever searchers do so, *agile*. Note that as long as we assume the speed of the fugitive to be unbounded, it is not meaningfully distinct from one being able to move at will. Recently, it was proved that the mixed search game against an agile and visible fugitive is monotone in the paper of this title [187]. The corresponding *mixed search number against an agile and visible fugitive* of a graph G, denoted $\mathrm{avms}(G)$, has been found equal to Cartesian tree product number [144, 79]. Furthermore, a search strategy can be derived from a newly introduced loose tree decomposition.

An intuitive approach is also proven correct in the case when the fugitive is invisible but *lazy* (or *inert*), that is, it does not move unless its capture is threatened [80]. Since the evader will not try to slip by pursuers while they are occupied

with searching another branch, they will eventually choose the correct branch, as is the case when the evader was visible. This result of equivalency of search numbers between an inert and visible fugitive was formally proven for the extension of node search model in [80]. Furthermore, the authors introduce a class of these problems by quantifying the speed of the fugitive in the amount of edges it can traverse when forced to move. When the speed is equal to 1, then the problem is monotone and the search number is equal to the width (linkage) of the graph. This parameter is interesting on its own [168], and thanks to this result the authors were able to approach it from a game-theoretic angle consistent with tree and path decompositions. For an arbitrary speed $s \in \{1, \ldots, n-1\}$ the authors notice that as long as the largest chordless cycle is at most $s+2$, the search number remains equal to the treewidth plus 1. In other words, restricting the speed of the fugitive does not help search such a graph.

In this paragraph we will outline the fruitful efforts to unify results for games with visible and invisible fugitive contained in [125]. Once again, we invoke the reader's intuition to imagine searchers prepared with a plan based on a tree decomposition. Additionally, they have an ability to query an oracle for information about whereabouts of the fugitive. Whenever they encounter a branch in their plan, the oracle's answer allows them to choose correctly. Their strategy is therefore not unlike the one in the case of a visible fugitive, despite their abilities being more restricted. The model, named *nondeterministic graph searching*, is related to a q-branched tree decompositions (and its treewidth), where, informally speaking, the parameter q refers to the maximum number of branches, and thus queries available to searchers, on the way from the root to a leaf in the tree underlying the decomposition. Note that 0-branched treewidth is equivalent to pathwidth, while for unbounded q it is equivalent to treewidth. As such, it offers a tool enabling a unified approach to both parameters. One example of a constructive use of this tool is the provided exact algorithm for computing q-branched treewidth for all $q \geq 0$, and thus also standard pathwidth and treewidth. It is exponential, but matched the performance of other developed algorithms for pathwidth (recall the NP-completeness of the standard problems). Furthermore, for any graph G of treewidth $\mathrm{tw}(G) = k$, the smallest number of queries such that G can be searched by the minimal amount of $k+1$ of searchers is at least $\log_2(\lceil \mathrm{pw}(G)/\mathrm{tw}(G) \rceil)$. This work was followed up in [184], which established that the problem is monotone, thus proving that the aforementioned algorithm computes not only decompositions but optimal non-deterministic search strategies.

1.3.5 Other layout parameters

In this short section we briefly mention a few results establishing connections of search numbers with other graph layout parameters that did not find their place in the above sections. Intuitively, a (monotone) search strategy can be thought of as a (linear) ordering of edges to be cleaned. Hence, it is not uncommon for these fields of research to intersect. [34] surveys relations between pathwidth, treewidth and parameters connected with linear orderings of vertices, such as: bandwidth,

cutwidth, modified cutwidth, vertex separation number, which naturally makes the analysis relevant to searching problems.

First, a simple remark that *interval thickness* coincides with the node search number was proven in [166]. Less straightforward connection exists with *topological bandwidth*, summarized in the article of the same name [181], and denoted $\mathrm{tb}(G)$. For any graph G, $\mathrm{ns}(G) \leq \mathrm{tb}(G)$, and its corollary $\mathrm{es}(G) \leq \mathrm{tb}(G)+1$. Next, for any G with the maximum vertex degree 3, its *modified cutwidth*, denoted $\mathrm{mcw}(G)$, is not greater than node search number minus 1 ($\mathrm{mcw}(G) \leq \mathrm{ns}(G)-1$). Furthermore, the authors claim that for graphs of arbitrary degree this result can be generalized to $\mathrm{mcw}(G) \leq \lfloor \Delta(G)/2 \rfloor \cdot \mathrm{ns}(G) - 1$ and mention a similar analysis for *cutwidth* and edge search number in [182][5].

Let us finish with an introduction of an idea which considers alternative measures of efficiency of graph searching. [126] introduced the *search cost* in node graph searching. Rather than taking a maximum number of searchers used at once (more formally, in their defined instance of a "step"), the authors propose taking a sum of numbers of searchers in each step. The search cost, denoted $\mathrm{sc}(G)$, is the minimum over such sums in strategies that search the graph. Computing $\mathrm{sc}(G)$ is related to finding the *profile* [25] of the graph and solving the *vertex separation sum problem*. Recall that the connection with pathwidth was established by its relation to the vertex separator (see Section 1.3.2).

Adaptation of the search cost parameter to the edge searching model was made in [87]; however, finding its respective layout parameter was left as an open problem. Nonetheless, a transformation from the problem of finding minimum cost edge search strategies to the problem of minimizing the cost in the node search model was provided. Moreover, efforts towards finding the *minimum edge search cost*, denoted $\mathrm{esc}(G)$, may lead towards using a greater number of searchers. In essence, it is not possible to optimize for both $\mathrm{es}(G)$ and $\mathrm{esc}(G)$ for arbitrary graphs. Two ways to reconcile such pairs of problems are: seeking specific classes of graphs where the relevant parameters coincide and two-criteria optimization. This latter concept was explored in [92], concerned with, as the title suggests, trade-offs between width-like (pathwidth, treewidth) and their respective fill-like (profile, fill-in) graph parameters[6].

1.4 Fast and wide

The above sections were concerned with the classical approach to searching, focusing mostly on feasibility of formulating a search strategy with given resources and conditions. Nonetheless, once conditions for feasibility are established, it gives way for new questions to arise, among which one of the most natural ones is to determine a "time" required to execute a search strategy. However, the formal definition

[5] The authors of [181] use the phrase "graph of degree x" and notation $deg(G)$, which we believe refers to the maximum vertex of graph G.

[6] The authors note that they are not the first to notice the existence of the trade-offs, citing [31, 169].

of time required to search a graph is not immediately obvious. The approaches in literature differ in expressing it as a graph parameter and bounding the number of moves executed by a strategy.

This branch of research received comparatively little attention early, until the introduction of fast searching in [98]. A tongue in cheek explanation of this indifference might be the pervasive insistence that polynomial solutions are efficient. In this view, if a model is monotone, it is usually easy to establish a satisfactory upper bound on the number of moves.

1.4.1 One giant step for searcher-kind

We open this section with a general motivation: if the goal is to minimize the time of a search, there is a good chance that we are ready to spare no expense to achieve this goal. This is the premise of the work of Chang in 1991 [53], who investigated the possibility of searching graphs in Parsons' edge searching model in a single *step*. In a step each searcher is allowed to execute a single action. Thus, searchers which slide along their edges simultaneously count as moving in a single step. The difficulty comes from the fact that the fugitive can hide between searchers while one is moving towards and the other away from a vertex. Note that this problem is trivial in the node and mixed searching models — one can simply put a searcher on every node.

The lower bound on the number of searchers necessary to perform such a feat in the edge searching model is m, and a graph can be searched in a single step with m searchers if and only if it does not contain an odd cycle as its subgraph. Nonetheless, determining what number of additional searchers is necessary is NP-Hard [53]. Later it was proven to be equivalent to the maximum two independent set problem [152]. The research was continued on a generalized model [154, 153], where a few weighted versions were investigated, although it did not garner significant attention. An open question that, to the best of my knowledge, was not tackled in the edge searching model is the generalization for a given number of steps s: what is the minimum number of searchers needed to search a given graph in s steps.

1.4.2 A long path

The task of defining general node searching time parameters formally was undertaken in the new millennium [44] and resulted in characterizations of the length of a strategy in terms of path decomposition, interval graphs, and vertex separation. The problem of finding both of these parameters was proved to be in NP thanks to the monotonicity of the underlying node searching problem (keep this specific variant in mind for the purpose of a comparison with the next section). The authors identify examples of classes of graphs such that even a single additional searcher yields time improved twofold. Moreover, they provide an example such that at least a specific amount of extra searchers is required to achieve any improved result at all.

Let us build upon the intuition developed in the previous section and the definition of pathwidth. Recall the sequence of subsets of vertices $X_1, \ldots, X_r$ in the definition. While the size of the biggest subset, henceforth denoted k, corresponds to the width of the path, let its length be expressed in terms of the number of these subsets, i.e. r. Thus the pathwidth problem can be generalized to include r as the second criterion. Minimization of this parameter was considered in [89], which was concerned with minimum-length path decompositions. This generalized problem is NP-Complete for general graphs for any fixed $k \geq 4$, and also for any fixed $r \geq 2$ [89]. Note that the original single criterion pathwidth problem was fixed parameter tractable in regards to k. When $k < 4$, a polynomial time algorithm is provided. Furthermore, the problem for connected graphs is NP-Complete for any fixed $k \geq 5$, and polynomial for any $k \leq 3$. To the best of my knowledge, the open question of the remaining case $k = 4$ remains unresolved.

Similar results were obtained for the generalized treewidth problem [173]. It was proven that the problem is NP-complete for any fixed $k \geq 4$ and polynomial for $k \leq 2$. In the missing case of $k = 3$ the problem was found to be polynomial for trees and 2-connected outerplanar graphs. We note that an effort was put into optimization of algorithms for these problems, resulting in subexponential time algorithms for $k \geq 4$ in both cases [37].

1.4.3 Only once

In contrast to the previous approach, which stressed the ability of searcher to work in parallel, [98] defined fast searching by modifying Parson's edge search model by forbidding the type of move which removes a searcher from a graph and introducing the condition of traversing each edge only once during the execution of the entire strategy. Thus, the problem trivially has number of moves linear in the number of edges and is monotone. Let $\mathrm{fsn}(G)$ denote the *fast search number*, i.e. the number of searchers required to execute such a strategy in a graph G. The initial exploration of the problem showed that the problem differed substantially from not only the basic edge searching, from which it was derived, but most other searching problems as well. Unlike them, it is not minor closed, which can be illustrated simply by providing an example of a graph G and its subgraph[7] G' such that $\mathrm{fsn}(G) < \mathrm{fsn}(G')$ (see Figure 1.3).

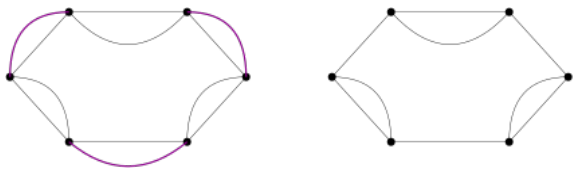


Figure 1.3: The graph G on the left and its subgraph G' on the right such that $\mathrm{fsn}(G) = 3$ and $\mathrm{fsn}(G') = 5$ [98].

[7] To be precise it is also a proof of a stronger property of the problem not being closed under subgraphs.

The first results were a linear time algorithm for trees [98] and the characterization of bipartite graphs in terms of their fast search numbers. Further results for specific classes of graphs include: grids, cubic and Halin graphs [217], complete k-partite graphs [232], Cartesian products of certain classes of graphs [231]. Furthermore, as conjectured by the parameter's creators, the problem of determining the fast search number $\mathrm{fsn}(G)$ for an arbitrary graph G turned out to be NP-hard, as was shown in [86] and [235] using different techniques. The former of the two articles containing the aforementioned proof of NP-completeness is concerned with the characterization of graphs F such that $\mathrm{fsn}(F) \leq 3$.

The latter article investigated connection of fast searching to the established models in more detail. Due to the restrictions placed on fast searching, finding $\mathrm{fsn}(G)$ for a graph G is different than minimizing the number of moves[8] in the edge searching model, and this distinction is considered in [235] with the introduction of the *fast edge search time*, denote $\mathrm{fet}(G)$, and the corresponding *fast edge search number*, denoted $\mathrm{fesn}(G)$. The former parameter denotes this minimal number of moves and the latter is equal to the number of searchers needed to clean the graph in $\mathrm{fet}(G)$ moves. The author also draws connections of their model to not only fast but also node searching and pathwidth. Finally, the author provides an approximation algorithm which computes a fast search strategy. Note that, while $\mathrm{fesn}(G)$ and $\mathrm{fsn}(G)$ are different, every fast search strategy is also an edge search strategy, thus an approximation can be derived for both from the proposed algorithm.

One more model obtained by introduction of an additional requirement to an already established model is fast-mixed searching proposed in [236] with its corresponding *fast-mixed search number*, denoted $\mathrm{fms}(G)$ for a graph G. The author has proved the NP-completness of the problem, its relation with the induced-path cover problem (see [172]) and investigated a few special classes of graphs. Every fast-mixed search strategy is also a mixed search strategy, where the searchers obey an additional rule — they can be placed and move only on occupied vertices and edges, respectively.

As a closing remark of this section we refer a reader who is interested in exploring a few further examples concerned with time of search in pursuit evasion models to [2].

1.5 Breaking the monotony

Consider an application of an edge search strategy by a team of entities in practice. There is a hidden cost associated with its execution since the actions allow for removing and placing an entity on an arbitrary vertex, which we have to circumvent by traveling along additional edges. To our dismay and horror, many models discussed so far share this property, which makes the stated purpose of searching problems secondary to their value as a tool for exploring, undoubtedly

[8] While the article uses the term "step", we decided to replace it with "move" to keep consistency and avoid confusion with the single step searching discussed earlier.

very useful, graph properties. Although this distinction was noticed as early as 1980s by Megiddo at al. [186] in their discussion of length of a search strategy, this problem was first addressed in [11] and shortly followed in [12][9] in 2003. The latter work opens with a vivid story, narrated in an irreverent tone, set in a library where jumping over the bookshelves is frowned upon. The authors introduced two, seemingly innocuous, properties that can be required of search strategies, summarized as follows:

- a search strategy is *internal* if searchers cannot be removed from a graph (e.g. in internal edge searching model they can be placed and henceforth perform only sliding actions). This ensures that searchers cannot "jump";
- a search strategy is *connected* if the set of clean edges is always connected.

These two properties together form the basis of *contiguous* searching, although different authors may choose to highlight only one of them.

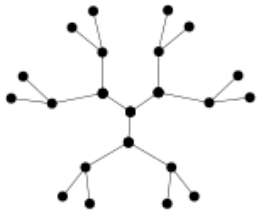
Recall that a search strategy is monotone if no recontamination (a previously clean edge becomes contaminated) occurs during its execution. If it is unknown whether there exists an optimal monotone strategy for all instances of a researched problem, then it may be beneficial to restrict the scope of the problem to only allow such strategies, which are easier to analyze. Note that these properties need not be explicit. For example, the conditions imposed on searchers in the fast searching model imply that any strategy obtained in this model is both internal and monotone. Conversely, edge searching required a non-trivial proof that the problem is monotone. Furthermore, these requirements can be combined in order to create new search problems. Recall that for a graph G, $\mathrm{es}(G)$ denotes its edge search number. Listed below are problems obtained by modifying it and their respective search numbers:

- internal edge searching, $\mathrm{ies}(G)$,
- monotone edge searching, $\mathrm{mes}(G)$,
- monotone internal edge searching, $\mathrm{mies}(G)$,
- connected edge searching, $\mathrm{cs}(G)$,
- internal connected edge searching, $\mathrm{ices}(G)$,
- monotone connected edge searching, $\mathrm{mces}(G)$,
- monotone internal connected edge searching, $\mathrm{mices}(G)$.

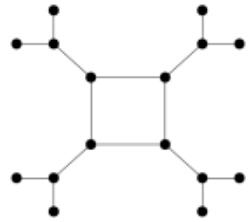
The following chain of relations was established in [12]:

$$\mathrm{es}(G) = \mathrm{ies}(G) = \mathrm{mes}(G) \leq \mathrm{mies}(G) \leq \mathrm{cs}(G) = \mathrm{ices}(G) \leq \mathrm{mces}(G) = \mathrm{mices}(G),$$

where G is an arbitrary graph. Furthermore, a similar analysis was conducted for trees. See Tables 1.2 and 1.3 for a brief summary.



(a) The graph T such that $\text{mies}(T) = 4$ and $\text{ies}(T) = 3$ (compare with Figure 1.1) [12]



(b) The graph G such that $\text{ies}(G) = 3$ and $\text{mies}(G) = 4$ [237].

Figure 1.4: Examples of graphs that show that internal searching is not monotone

Overview for general graphs G	
Relation	Proof
$\text{es}(G) = \text{ies}(G)$	Trivial[1].
$\text{es}(G) = \text{mes}(G)$	Edge searching is monotone [171, 24].
$\text{ies}(G) \leq \text{mies}(G)$	Internal searching is not monotone; see Figure 1.4 [12, 237][3].
$\text{mies}(G) \leq \text{cs}(G)$	[12][3]
$\text{cs}(G) = \text{ices}(G)$	Trivial[2].
$\text{mces}(G) = \text{mices}(G)$	Trivial[2].
$\text{cs}(G) \leq \text{mces}(G)$	Connected searching is not monotone [237]
$\text{mces}(G)/\text{es}(G) = O(\log n)$	[10]
$\text{cs}(G)/\text{es}(G) \leq \log n + 1$	Using connected treewidth in 2006 [130].
$\text{cs}(G) \leq 2\text{es}(G) + 3$	Using connected pathwidth in 2012 [85].

Table 1.2: 1. An internal strategy requires additional moves to manually slide searchers that would be removed and placed again in a non-internal strategy.
2. Searchers move freely in the connected component.
3. The difference might be arbitrarily large [237].

Specifically, $\text{ies}(G) \leq \text{mies}(G)$ has simple examples of graphs where the inequality is strict, which reveals that monotonicity of edge searching is the product of searchers being able to jump around the environment. It is not surprising that this fact was emphasised by the researchers who were looking at the problem from the perspective of mobile agents earlier [11]. Since this is the first time agents are mentioned and the topic requires an introduction of a considerable size, let us postpone the discussion of this aspect of their work until Chapter 2. The status of the last inequality, namely $\text{cs}(G) \leq \text{mces}(G)$, has proven to be much more interesting, as we will see in the next section.

[9]Note that [13] is cited more frequently, but one of the proofs omitted there appears in [12].

Overview for trees T	
Relation	Proof
$\mathrm{es}(T) \leq \mathrm{cs}(T)$	By example, see Figure 1.4[1] [12].
$\mathrm{es}(T) = \mathrm{mes}(T) = \mathrm{ies}(T) \leq \mathrm{mies}(T)$	Same as for general graphs.
$\mathrm{mies}(T) = \mathrm{cs}(T)$	[12].[10]
$\mathrm{cs}(T) = \mathrm{ices}(T) = \mathrm{mces}(T) = \mathrm{mices}(T)$	Connected searching is monotone for trees [11, 10].
$\mathrm{cs}(T) < 2\mathrm{es}(T) - 2$	[12, 10]

Table 1.3: 1. Note that an alternative proof can be obtained by combining the second and third row.

1.5.1 Connected graph searching is hard

Note that if $\mathrm{cs}(G) \leq \mathrm{mces}(G)$ can be strict, then it implies that connected graph searching is not monotone. While in [13] it was left as an open question, the proof of that claim appeared independently in 2009 [237] by different authors. The results from [13] on both internal and connected searching were expanded, although we will focus on the latter. The authors construct graphs using cliques in such a way that their search numbers can be easily determined and end up with an example of a family of graphs W_k, $k \geq 1$, such that $\mathrm{cs}(W_k) = 280k + 1$ and $\mathrm{mces}(W_k) = 290k$. While they do not claim that it is the smallest example, as they remark that the construction can be also scaled down, it is in a stark contrast with the simple examples of graphs provided for internal searching.

Furthermore, it is shown that connected and monotone internal searching problems are not minor closed, that is, there exists graphs G, H such that H is a minor of G and $\mathrm{cs}(G) < \mathrm{cs}(H)$, echoing the conclusions of [13]. In fact, a subgraph may require arbitrarily many more searchers to clean it than its supergraph. This property, in conjunction with the fact that non-monotone strategies do not have a natural polynomial bound on the number of moves, has proven to be an obstacle that prevents application of many techniques developed for analysis of other searching problems. Nonetheless, a specific case of graphs searchable with 2 searchers in the connected, both monotone and non-monotone, mixed model were characterized using the same set of 177 obstructions [20]. Note that in the connected variant there is no guarantee that such a set of finite size exists for each number of searchers, so extending this line of research into a general result is not very promising. Despite all the efforts, to this day it is not know whether determining $\mathrm{cs}(G)$ is in NP[11].

The efforts were then focused on the "price of connectivity" defined as a ratio of connected search number $\mathrm{cs}(G)$ to $\mathrm{es}(G)$ for a graph G, which was known not to exceed a factor of 2 for trees [12]. Progress was made thanks to the concept of connected treewidth [130], further applied to chordal graphs [193]. Ultimately, the conjecture that $\mathrm{cs}(G)/\mathrm{es}(G) \leq 2 + o(1)$ for arbitrary graphs was proven true in [85], while [10] was in review, using the notion of connected pathwidth. Moreover,

[11] There exists an example of a searching problem that is both non-monotone and in NP, namely the internal strong searching of directed graphs [234]. Since the topic of searching digraphs is outside of the scope of this dissertation, we refer an interested reader to the third chapter of [40].

the factor 2 is tight, thereby closing the posed open question. Additionally, a method of converting a given search strategy using k searchers into a monotone and connected one using $2k + 3$ searchers was provided.

Given the extensive literature on the topic of fixed-parameter tractability of the pathwidth problem, it was reasonable to ask whether the same holds for its connected sibling. Due to the fact that this parameter is not closed under minors, the techniques used for the regular pathwidth do not apply in a straightforward fashion. It is, however, closed under contractions. For a summary of these and further observations on the difficulties associated with the problem, see the introduction of [160]. As a start of interest in this investigation one can point to the GRASTA 2017 workshop, where among open questions [124], the following was raised: is it possible to verify in polynomial time, whether, for a fixed constant k, the connected pathwidth of a given graph is at most k. This was the stated motivation behind [91], where an algorithm solving this problem in time $f(k) \cdot n^{O(k^2)}$ was provided. The question of fixed-parameter tractability was settled in [160], with the introduction of the algorithm of complexity $n \cdot 2^{O(k^2)}$. Note that these results do not resolve the open question of whether the connected graph searching problem is NP-Complete because connected pathwidth is, by definition, concerned with monotone strategies.

Connected searching with a visible fugitive was considered in [131], where it was shown to be non-monotone. Furthermore, the logarithmic $\Theta(\log n)$ bound on the price of connectivity of visible searching was proven to be tight.

1.5.2 Connected tree searching is easy

In this section we will consider connected searching of classes of graphs such that the problem becomes polynomial, with trees serving as the primary example. Despite the upbeat title, let us start with a cautionary tale. A generalized version of the problem of connected searching of trees, in which an edge or a node of a tree can require more than one searcher was claimed to be solvable using the modified, but still linear, algorithm for the unweighted case [11]. This assertion was proven wrong in [83], as the weighted version of problem turned out to be NP-Complete. The continuation of this line of research resulted in a polynomial time 3-approximation algorithm for connected searching of weighted trees [84]. For those especially inquisitive we included Tables 1.2 and 1.3 where one can find references to the proofs of the other presented claims. Thus, for a reader wishing to familiarize themselves further with connected searching, perhaps because of a piqued interest in the open question of NP membership, we discourage following the chronological order and recommend starting with [10] instead. This work contains republished results from [11, 12] supplemented with a commentary on the works that appeared in the decade between publications.

As expected in a searching problem, restricting the environment to tree graphs has proven to reduce its difficulty tremendously, in computation as well as analysis.

The following results are provided in [12]:

$$\mathrm{es}(T) = \mathrm{ies}(T) = \mathrm{mes}(T) \leq \mathrm{mies}(T) = \mathrm{cs}(T) = \mathrm{ices}(T) = \mathrm{mces}(T) = \mathrm{mices}(T);$$

where T is an arbitrary tree (see Table 1.3 for a brief summary). Thus, despite the apparent plethora of possibilities only two values of considered search numbers do not coincide for trees. While many of these results follow from the ones for general graphs, a notable distinction is $\mathrm{cs}(T) = \mathrm{mces}(T)$ — connected searching is monotone for trees [10]. Furthermore, it is minor closed and, unlike the classical edge searching of trees, for each k obstruction set $\mathcal{T}'_k$ that characterizes trees such that their connected search number does not exceed k consists of one element. It is easy to check that $\mathrm{es}(D) = \mathrm{cs}(D)$, for each $D \in \mathcal{T}_1 \cup \mathcal{T}_2 \cup \mathcal{T}_3$ in Figure 1.1. And indeed $\mathcal{T}'_k = \mathcal{T}_k$, $k \in \{1, 2, 3\}$. For $k \geq 4$, $\mathcal{T}'_k$ contains the star with 3 edges, each endpoint of valency 1 connected to the root of a copy of a full binary tree of height $k - 2$. See Figure 1.4a for the single tree in $\mathcal{T}'_4$. Alternatively, one can describe trees such that $k + 1$ searchers suffice as k-caterpillars. A succinct corollary to this result is $\mathrm{mces}(T) = \log n$ for binary trees. This very simple characterizations lead to an algorithm that computes a connected search strategy for arbitrary T in linear time (given a starting vertex, thus even an exhaustive search over all possible starting positions guarantees the complexity of $O(n^2)$).

We will revisit the topic of connected searching during the discussion of decontamination problems in the next chapter.

1.6 In search of oddities

While many results in the field of graph searching are highly connected, some search problems are not easily classified. The author realizes that the concept of heterogeneous graph searching, one of the main subjects of this dissertation, is likely to belong to this category as well. However, due to the idea being related with problems in other fields of research its discussion is postponed until Section 3.2.

A seemingly arbitrary, although we will see that not to be the case in Section 2.2.7, condition that two searchers can never occupy the same node gave rise to the *exclusive searching* model [29]. This model differs considerably from internal, mixed searching, on which it is based. Not only it is non-monotone but also it is not closed under a subgraph. A simple analysis of a star graph shows that exclusive search number is related to the degrees of graph's vertices. Furthermore, [183] shows that its computational complexity is not related to that of pathwidth, which was not proved for any other searching problem before. This was achieved by comparison of monotone exclusive searching, which is NP-complete in split graphs and polynomial in starlike graphs, with the complexity of the pathwidth problem, which is polynomial and NP-hard for these classes, respectively. In general, exclusive searching is NP-hard in planar graphs with bounded maximum degree and it is not known whether it is NP-complete for arbitrary graphs.

LIFO-search, as defined in [137], is a variant of the node graph searching problem where only the most recently placed, or last in, searcher can be taken off the

graph, i.e. it is first to be out. Thus, since the rules of the node searching variant forbid performing sliding moves, it is not possible to move other searchers while it is on the graph. This variant is monotone and the minimum number of required searchers does not vary whether the fugitive is visible or invisible. The minimum number of searchers required is equal to *treedepth*, a parameter that can be conceptualized as a measure of how much a given graph resembles a star. While the problem may seem niche from the perspective of graph searching, the authors note that finding the treedepth of a graph is related to solving the vertex ranking problem, the ordered colouring problem, finding the minimum-height of an elimination tree of a graph, and it appears in theory of graph classes of bounded expansion [191]. The same work also investigates LIFO-search on digraphs and its relation to cycle-rank.

Let us close this overview of less known models with a mention of a work [82] describing a non-monotone problem which does not use the concept of search number but focuses on minimizing the maximum number of steps in which a vertex is occupied by a searcher. This property, named a maximum vertex occupation time, was studied in a model with an inert fugitive and may differ arbitrarily when only monotone strategies are considered.

Chapter 2

Mobile agents

We have only mentioned agents and distributed environments in the previous chapter when discussing the origin of the internal and connected searching models [11], leaving the reader with a promise of expanding on the topic later. This is the chapter where we fulfill this promise.

2.1 Agents and their environment

With the introduction of connected and internal searching, a lot more attention was paid to the possibility of implementing an algorithm in an existing network. For example, in an internal search problem searchers are prohibited from jumping, thus ensuring that the actions specified by a search strategy could be executed by an entity moving in an environment which the given graph models. This approach of giving some individuality to entities, further called *agents*, is not uncommon in modelling pursuit-evasion, rendezvous and exploration problems[1]. Furthermore, when we wish to emphasise agents' ability to change position, we will refer to them as *mobile*.

If while constructing a solution for a given problem each agent has access to the complete information, then we will call the algorithm (or problem) *centralized*. Note that all of the search problems discussed so far assumed the possibility of executing a centralized algorithm. On the contrary, if we assume a presence of some local knowledge specific to each agent which is not trivially shared, then we will call the problem *distributed*. we refer to such agents, acting on their local information in a distributed environment, as *autonomous*. Note that the requirement that sharing knowledge is not trivial ensures that agents cannot coordinate to compute a solution identical to a one created by a centralized algorithm without incurring a meaningful cost. When discussing instructions that govern an agent's behaviour,

[1]The term "agent" can also be found in other disciplines and software development. Our use refers broadly to entities encountered in graph problems. Hence, it encompasses cops in helicopters [21], not employees of MI5.

we will refer to its *protocol*. While protocols need not be distinct between agents, nonetheless their execution depends on local information. In fact, by specifying a role of an agent as an input, all agents can be made to follow the same protocol [210].

Depending on the specific model, agents may have different methods of communication available to them. A very restrictive model could allow for an exchange of information only when two agents share position at the same time. Relaxing the latter requirement gives us a model with whiteboards, where agents can write information locally for others to read later. Note that a limit of available memory, either on whiteboards, or available to agents themselves, is another field of optimization.

When some of the knowledge of agents is local, the execution of an algorithm could appear random due to their choices being based on arbitrary description of their environments. In the case of problems where the goal is optimizing for the worst case scenario, a useful concept for analysis of such problems is an *adversary*. When an agent obtains new information, e.g. a label of a visited node or even its own label during an initialization, then it is decided by the adversary. Moreover, in such a way that it hinders reaching the goal to the greatest extent. Note that as long as the adversary follows the restrictions given by the problem and all of its decisions are consistent with the data made available to agents so far, there is no need for a predefined underlying world to exist, only bits unveiled by the adversary ad hoc[2].

In literature one can also find the distinction between centralized and distributed problems where the adversary is given such power under the names *off-line* and *on-line*, respectively. Since the research of distributed problems is very broad, the exact line between on-line and off-line problems may vary. To give a concrete, relevant example, the case of searching an unknown graph would be an *on-line* problem for [197], while [43] uses the terms online and distributed interchangeably. Naturally, it is expected that when a problem is transformed into a distributed model the efficiency of algorithms drops. The difference in the performance or use of resources between the respective on-line and off-line versions of the problem is referred to as *competitive ratio* (the concept is similar to the price of connectivity outlined in the previous chapter). Due to these inconsistencies in the used terminology, we defer the introduction of the formal definitions of the concepts used in our results to their specific chapters.

When the graph is *a priori* unknown to agents, they are usually granted some local, i.e. available to an agent visiting a node, information that allows them to navigate their environment. To discuss the works in further sections, we introduce here a notion of a *port*. It is, intuitively, an entry or exit point of an edge from the point of view of an agent. Let us assume that an agent on a node u can distinguish, at least, labels of each port, denoted by distinct integers from 1 to the degree of u. An agent knows the port by which it arrived to a node and can choose which port it will use to leave the node. Note that, in the absence

[2]See [192] for an old, radical application of this idea in a distant field.

of additional information, numbering of ports alone is insufficient to determine whether a particular node has been visited by an agent. One needs only a simple a example of a ring network, where every node has two ports, namely 1 and 2, to verify this observation. A further distinction is concerned with whether the numbering of ports is consistent for all agents, or the adversary can label the ports differently for each agent. To further elucidate this point, in the aforementioned example of a ring network a common labeling can be used to distinguish common 'clockwise' and 'anti-clockwise' directions for all agents.

Furthermore, the adversary has a particular trick that frequently has to be considered when crafting a model to research. If two or more agents follow the same protocol and find themselves in an environment that they perceive in the same fashion, then they can be made to perform actions that mirror each other. This can lead to a lack of guaranteed solution, as is commonly remarked in descriptions of rendezvous problems. For example rendezvous on a node is impossible in a graph consisting of a single edge in the absence of distinct labels on both agents and elements of their environment — the adversary can ensure that they always swap nodes, e.g. as remarked in [94]. If a property of a given problem allows the agents to differentiate their actions, then we say that it *breaks the symmetry*.

Another property that can be used to distinguish distributed environments is the perception of time by agents. A model is called *(fully) asynchronous* if the agents can rely only on the knowledge that their actions will take a finite amount of time[3]. If an agent is provided a local clock, its units of time are unknown. Again, the adversary can be used to govern the time of performance of each of agent's actions and each of their clocks. On the contrary a *(fully) synchronous* model is characterized by two properties: every local clock ticks simultaneously and the upper bound for the time of actions is known. These restrictions are sufficient to establish an external view of the system, where each action is known to be completed after one unit of time (a unitary delay) [210].

Models may impose additional restrictions, such as in the following Look-Compute-Move model, also known as CORDA. In this model each agent performs actions in asynchronous cycles of three consecutive phases:

1. Look - the agent takes a still snapshot of the environment and relevant information.

2. Compute - the agent decides its behaviour in the next phase.

3. Move - the agent executes an action of moving to an adjacent node or staying idle. Note the the information obtained in the first phase might be outdated at a discretion of the adversary.

As we will see in Section 2.2.7, it was featured in the context of perpetual graph searching [28]; however, it is also used more broadly, for example in rendezvous

[3]This definition is commonly amended by a remark that this assurance excludes the case of failures. In our review, it applies only to Section 2.2.5, and the absence of failures is assumed in all the other cases.

[123]. The complexity of computation involved in the second phase may be considered irrelevant. This is not an unusual occurrence when discussing distributed models, where it is often preferred to express the efficiency of a solution in terms of the number of actions or, in the case of synchronous models, *rounds* which consist of synchronized actions of all agents in a unit of time. A diligent reader might recall that similar measures were considered in Section 1.4 where the speed of searching was considered. In particular the concepts of a step and a round, where entities are allowed to execute actions simultaneously, would be analogous.

In the following sections, we will discuss some distributed problems before addressing the centralized algorithms that were described as part of the original research that constitute this thesis. Finally, as a closing remark for this introduction, we would be remiss not to acknowledge the two big branches of graph research using agents — exploration and rendezvous. Although we will not be covering these in detail, we refer the unsatisfied reader to the following surveys: [75] for the former, and [200] for the latter.

2.2 Decontamination

Let our first example to breach the gap between centralized and multi-agent problems be a distributed version of graph searching. A searcher is thus modeled as an agent (in this section the terms 'agent' and 'searcher' will be used interchangeably) with some, dependent on the specific model, degree of independence from its peers. Actions of the fugitive can be said to be controlled by the aforementioned adversary, which ensures that the proposed strategy needs to be able to handle the worst case scenario. Fortunately for our introductory problem, a description of the fugitive as an autonomous entity outside of an algorithm's control does not yield additional insight in our formulation of the problem. We recall that it is sufficient to distinguish between elements of a graph which the fugitive can possibly occupy, i.e. contaminated ones, and those that are sure to be free of its presence, i.e. clean. The title of this section adheres to the naming convention that distinguishes the distributed branch of graph searching research as *decontamination problems*, although this is certainly not a strict rule and the name is sometimes used in a broader context.

2.2.1 A second look at connectivity

Recall the connected internal variant of searching developed in [11] from Chapter 1.5. After its introduction it was followed with a discussion of connectivity and monotonicity of centralized searching algorithms. In this section we focus on the practical and distributed aspects of the model. The former is reflected by the strategy being internal, since mobile agents are rarely equipped with an ability to arbitrarily relocate at will. Additionally, all agents enter the environment at once and at a single node. In general, a node distinguished as a point of entry for all agents will be called their *homebase*. Furthermore, a distributed network environ-

ment gives rise to a concern of safety of searchers. In this regard, the connectivity ensures that communication between agents can follow clean edges, and thus cannot be tampered with. Note that the term contiguous can be encountered to mean internal connected searching (e.g. in [117]). And lastly, some of the edges might require more than one searcher to clean (represented by weights on edges). Unfortunately, as mentioned in the beginning of Section 1.5.2, the authors' algorithm for decontaminating weighted trees does not succeed in this regard.

Nonetheless, the algorithm for non-weighted trees provided in [11] has further practical advantages. It has time complexity $O(n)$ and the computed sequences of moves can be stored in $O(n)$ space. The authors further note that this is despite the fact that some trees may require at least $n \log n$ moves. Finally, if each node is a processor, a strategy can be computed in a distributed fashion using $O(n)$ messages, thanks to a possibility of assigning a certain common labeling of nodes.

2.2.2 Overview of properties

Let us now consider further adaptations of the baseline internal connected model outlined above and its studied variations. We begin with with a very brief and general overview of the works that share many assumptions in their models. It is followed by description of properties encountered in decontamination problems, which conclude with the assignment of works to the outlined properties. Section 2.2.3 readdresses the works used in the introduction below and discusses the obtained results. Further below one can find sections on more specific variants of the problems. An ever inquisitive reader can find tables with the detailed results in [194] and [197].

Landscape

The following graph topologies were considered by Flocchini et al.: hypercubes [112] (a preliminary version of [114]), meshes [117, 215], chordal rings and tori [113] ([111])[4]. This line of research was continued in the form of a refinement of an experimental approach in [119, 120]. PhD thesis of one of the coauthors [155] contains results for hypercubes, chordal rings and, additionally, butterflies. The leap towards purely theoretical results concerning the problem of decontamination of arbitrary graphs was made in [30] by a different team of researchers. Furthermore, pyramids were investigated in [214], star graphs in [159], Sierpiński graphs in [178], product networks in [158] and (unknown) partial grids were considered in [93], which later became a part of Osula's PhD thesis [197].

Properties

Let us investigate properties that may be encountered in decontamination models. As our point of departure let us assume the monotone internal connected searching

[4] A table summarizing results for these classes of graphs can be found in [194].

introduced in [11]. A property of the model [11] that has proven exceptionally unpopular almost to the point of uniqueness, at least in the world of graph searching, is providing an ability to communicate between nodes. Note that in this model the nodes themselves are treated as computational units which then can provide agents information sufficient to execute a strategy.

In the more popular decontamination models, in order to communicate agents relay on either meeting on the same node, where they can engage in a *face-to-face* communication, or leave messages on nodes. This latter method is facilitated by *whiteboards*, which represent some amount of local memory on each node common to all agents that visit it. Hence, any agent can read, erase, and write information visible to other agents, provided that they visit the same whiteboard. It is a standard practice to assume that agents can access a whiteboard on the rules of fair mutual exclusion. Furthermore, it may be assumed that the fugitive can corrupt any whiteboard it can reach. Conveniently, a search strategy being connected is sufficient to protect vital information under the properties introduced thus far.

Thus, let us jointly consider the following *local knowledge* available to any agent: the state of a node they are visiting at a given time, the presence of any other agents there and information available on the node's whiteboard (if they are included in the model). This basic set of assumptions about the knowledge accessible to an agent can be extended (or reduced to make the model stronger).

One such extension is the notion of *visibility*. For the sake of consistency between the discussed models, the property of d-visibility is defined as granting each agent an ability to obtain local knowledge of any node within distance at most d as if it was present on that node. Hence, 1-visibility allows agents to see the local knowledge of the currently occupied node and its neighbours. As it is the most commonly investigated variant of this property, further mentions of visibility without the prefix number refer to 1-visibility (0-visibility allows an agent to obtain only local knowledge). Its implementation can be achieved by sending single bit messages whenever a node is decontaminated to be stored on the node's neighbours [113, 214]. Besides the two aforementioned works, visibility is featured in [114, 155, 117, 178] and in the two experimental works [119, 120]. Moreover, a version of visibility with reduced local knowledge (only the status of neighbouring nodes is visible) is considered in [159, 158].

Recall that, since our baseline model is one of connected searching, it is natural to assume that every agent starts at a single position, their homebase. Note that when all agents start on a single node it is easy to ensure coordination of the whole team of agents by designating one agent (called a leader, coordinator, synchronizer) to facilitate communication between them. Thus, an algorithm can proceed as if agents operated with information shared globally, albeit at a cost of significant increase in the number of required moves. Observe that this approach works regardless of whether the environment is synchronous or asynchronous (a more precise remark on this property can be found in [93]). Nonetheless, the impact of allowing multiple homebases was studied using experimental approach in [119] and [120]. Naturally, the assumption of connectivity was dropped and, less intuitively, synchronicity was enforced.

Let us consider the measures of efficiency given the parameters discussed above: the number of moves, the number of rounds and, traditionally, the number of searchers. The order of this last parameter turns out to be largely independent of the specific choice of method of communication and synchronicity. As we have seen above, one additional searcher can transform an asynchronous strategy into a synchronous one and turn a model with long distance messages into one with a face-to-face communication. Conversely, a choice of a more restricted model in these respects might significantly inflate the number of moves or rounds. Let us note that the notion of time is not applicable to asynchronous models; hence, it is assumed that a traversal of a link takes one unit of time. This measure of idealized time will be used in place of number of rounds (steps) when synchronous and asynchronous models are compared in the next section.

An additional ability that may be given to agents in order to relieve this burden is *cloning*. In a model which admits it, an agent can create a fully functional copy of itself. The authors of specific works decide whether the cloned agent appears on the same node or is injected to one of the node's neighbours. This distinction often corresponds to whether the action counts towards the number of moves. Since this new agent counts as an additionally deployed searcher and minimizing the number of searchers is traditionally the primary goal, cloning is commonly paired with an ability to terminate an agent's existence. Cloning, or its unnamed equivalent, is featured in [114, 155, 158, 159, 158, 214] and, like visibility, the two experimental works [119, 120].

Other measures include space limitations such as the number of bits written on whiteboards or memory available to each agent. Further criteria are mostly model specific, such as the amount of additional information in [195] or spread of a virus in Section 2.2.5. Let us conclude with only a mention of a PhD thesis on best effort decontamination [204], where the goal is to decontaminate the greatest possible number of nodes when the number of searchers is insufficient to clean the whole graph.

2.2.3 Surveying the landscape

All works in this section share a similar set of considered properties in their models. We will use the term *local* or *basic* to mean the strongest model, i.e. asynchronicity, face-to-face and whiteboard communication, no cloning and no visibility (0-visibility). Moreover, a distinction between node and edge variants of the monotone internal connected searching is made[5]. Results will be discussed in terms of improvements achieved by relaxing these conditions through introduction of 1-visibility (shortened simply to visibility), cloning and synchronicity.

[5]In this section node and edge variant will always refer to monotone internal connected searching unless stated otherwise.

Hypercubes

Let us begin with the results regarding hypercubes [114, 155]. In all protocols agents use $O(\log n)$ bits of memory. The amount of the same order is used on whiteboards. The results for the local asynchronous edge and node connected variants of the model coincide. $\Theta(n/\sqrt{\log n})$ searchers are required, which make $O(n \log n)$ moves in the time of the same order. First, let us consider node variant where using cloning alone the researchers were unable to obtain a significant improvement, even in the latest iteration of the results [114]. Further introduction of synchronicity allowed to reduce the number of moves to the optimal $n-1$ executed in $\log n$ time at the cost of increasing the number of searchers to $n/2$. One can replace both of these properties with 1-visibility in order to obtain a strategy that uses the same number of searchers, $\Theta(\log n)$ time, yet asymptotically the same number of moves as the local strategy, i.e. $O(n \log n)$. Further introduction of cloning reduces this last value to $\Theta(n)$ moves. It is noted that further introduction of synchronicity does not appear to improve these results.

In the edge searching variant introduction of cloning alone reduces the number of moves and time to their optimal values of $(n \log n)/2$ and $\log n$, respectively. Thew new strategy uses $O(n)$ agents, and it is not known if synchronicity allows to reduce this number further.

Meshes

Next, we consider 2-dimensional meshes, described unambiguously in terms of the sizes of the 2 dimensions: $n_1 \times n_2$, $n_1 \leq n2$, investigated in [117]. Observe that $N = |V| = n_1 n_2$ and $m = |E| = 2N - n_1 - n_2$. The four directions are assumed to be common to all agents and the agents start in a corner.

The universal lower bound on the parameters is established in the basic, as defined above, model, to be n_1 agents, $n_1 + n_2 - 2$ time and N moves. To obtain constructive upper bound results, either an agent enforcing an order of moves (*synchronizer*) or 1-visibility was assumed, leading to 4 models once the divide between edge and node searching is included. Note that the presence of the synchronizer does not imply synchronicity. The two bounds match only in the case of node searching with assumed visibility for time and number of agents. The number of moves is significantly higher and expressed as $(n_1^2 + 2N - 3n_1)/2$. The protocols designed for the 3 remaining cases use one additional agent. Since the times and number of moves differ slightly in each case, let us only summarize them. In the remaining 3 cases the time is of order $O(N)$, and the number of moves in all cases is $O(n_1^2 + N) = O(N)$. Therefore, the bounds on the number of agents and moves are asymptotically tight. Moreover, both agents and whiteboards use at most $O(\log n_1)$ bits of memory.

We further mention a case of hexagonal meshes, i.e. such that each cell contains an additional diagonal, investigated in [122]. All diagonals are slanted in the same direction and are common to all agents. The considered cases in this frequently omitted work correspond to these of the node searching variant in [117]. Moreover,

the results for the number of agents are the same and these for the number of moves and time are asymptotically the same as in the aforementioned work.

Tori

From meshes let us proceed to their toroidal siblings, researched only in the node variant in [113]. Using the same notation as in the paragraphs above, we obtain $N = |V| = n_1 n_2$ and $m = |E| = 2N$ in a 2-dimensional torus. Moreover, this notation is extended to d dimensional toroidal meshes in the form of $n_1 \times n_2 \times \ldots \times n_d$ such that $2 \leq n_1 \leq n_2 \leq \ldots \leq n_d$.

The results for a 2-dimensional torus with added visibility are optimal: $2n_1$ agents, $\left\lceil \frac{(n_2-2)}{2} \right\rceil$ time and $N - 2n_1$ moves are necessary and sufficient. In the local model, one more agent is shown to be necessary. Therefore, it is indeed shown that the 1-visibility model is more powerful than the local model. Recall that in the previous research the gaps between lower and upper bounds for comparable parameters was not closed. Further note that the authors were unable to establish the optimality of their upper bounds on the time and number moves in this case. The latter is $2N - 4n_1 - 1$, while the former matches the number of moves with visibility ($N - 2n_1$). Note that this reflects the approach of an enforcement of an order of moves among agents by a designated coordinator which enforces "synchronicity". Intuitively, the cleaning strategy makes the 2 sets of agents move in the opposite directions along the dimension of lesser size (as stated in [113], it is a generalization of a strategy for a 1-dimensional torus, i.e. a ring).

Moreover, this strategy can be generalized for d dimensions, although no definite claims about its optimality are made, only a conjecture of such. After the generalization, the number of agents become $2N/n_d$ and $2N/n_d + 1$ for the case with and without visibility, respectively. When comparing the tables with results for 2 and d dimensions in [113], note that in the expressions of complexities the size of the lesser of the 2 dimensions (here denoted n_1) is replaced with N/n_d (note that when $d = 2$, then $N/n_2 = n_1$). Similarly, the size of the greater of the two (n_2) is generalized to become the size of the greatest among d dimensions (or simply $n_2 = n_d$ for $d = 2$).

Chordal rings

Results for chordal rings are split between the familiar [155] and [113]. Note that the research concerns fully symmetrical chordal rings, described as $C(\langle d_1 = 1, d_2, \ldots, d_l \rangle)$. An instance of such a class of graphs is a ring with additional $l - 1$ edges, shortcuts, between every pair of nodes in distances given by $d_j, j \in \{2, \ldots l-1\}$. Hence, each node has the same number of chords, is of a degree of $2l$ to be precise. Without loss of generality assume that all $d_i, i \in \{1, \ldots l\}$ are given in an increasing order (for the sake of consistency d_l will always refer to the longest link) and $d_l < \lfloor n/2 \rfloor$. Furthermore, the agents have a sense of direction.

Let us address [155] first. We begin with a general statement that a constructive upper bound of at least $2d_l + 1$ agents has been established for all the considered

variants (the only exception can be found in [113]). The edge searching variant was considered in the basic model and with synchronicity. Interestingly, this is the only variant where the efficiency of the algorithm depends on the number of chords — $2ln+2d_l^2-2d_l+2 = O(ln+d_l^2) = O(n^2)$ moves and $2ln-d_l+2 = O(n^2)$ units of time are needed. It is noted that synchronicity does not alleviate the issue of the sequential nature of the strategy and its synchronous adaptation would still take no less than twice the number of edges units of time.

A special case when $C(\langle 1, 2, \ldots, p, d_l\rangle)$ (lengths of chords are given by consecutive numbers from 1 to p and an additional, greatest, d_l) is called a chorded ring. The protocol for decontaminating a given chorded ring is considered in the synchronous edge variant. It works in n units of time and uses $2d_l + (p^2+p)/2 = O(d_l+p^2)$ agents and $O(pn+d_l^2+p^3)$ moves. Let us assure a scrupulous reader that, even when only the big O notation is used in this section, the exact polynomials can be found in [155].

In the basic node variant adaptation of the edge search strategy can be performed in $O(n+2d_l^2)$ moves and takes $4n-4d_l$ time. Adding synchronicity in this case does not change the asymptotic complexities, although it indeed reduces the needed time $(n+d_l-1)$. Asymptotic improvement in the number of moves can be achieved by adding the cloning ability which reduces their number to merely $2d_l - 1 = O(d_l)$.

Next, recall that [113] is concerned only with the node variant. It places further limitation on d_l, namely it assumes that $4 \leq d_l \leq \sqrt{n}$. The provided protocols use $2d_l + 1$ agents in the local variant and $2d_l$ when visibility is granted and, in contrast to [155], these values are proven to be optimal. The numbers of moves in their respective cases are $4n - 6d_l - 1$ and $n - 2d_l$, and the latter is also optimal. The protocol in the local model takes $3n - 4d_l - 1$ time units. When visibility is allowed the time becomes dependent on the length of two longest cords. Instead of providing the exact expression, let us focus on the intuition that increasing this distance allows to reduce the needed time through a greater parallelization of moves. Let us conclude with a remark on the asymptotic similarity between the expressions of time and number of moves, which take the form of $O(n-d_l) = O(n)$. Hence, the focus on constants appears justified, especially when exact optimal values are reached.

Butterflies

Before we leave Huang's PhD thesis [155] let us introduce their results for d-dimensional butterflies. The d-dimensional butterfly has $n = 2^d(d+1)$ nodes and $m = 2d^{d+1}$ edges. Let us further remark, without much explication, that it is stated that one can fold a hypercube into it and the structure of the d-dimensional butterfly is recursive. This last property is exploited in the design of the algorithms.

In the local variant, using $n/(d+1)$ searchers admits a protocol that executes $O(n \log n)$ moves in $3d-1$ time. Cloning alone reduces the total number of moves to m while the time and number of agents remain the same. These results hold for both node and edge variants, with a small note that, trivially, m total moves is the

optimal number of moves for edge searching. Further introduction of synchronicity in addition to cloning has no effect on the devised protocols.

Sierpiński graphs

The research within the established node searching variant of the basic model is continued in [178], a work of a lone coauthor of some the previous research, which investigates Sierpiński graps. This class can be though of as a graph representation of Sierpiński triangle fractal (the names Sierpiński gasket or sieve are also used), where d denotes how many iterations are used in construction. Thus, d-th iteration Sierpiński graph has $m = 3^d$ edges and $n = (3^d + 3)/2$ vertices[6].

The basic node variant is considered, then it is augmented with the visibility property. In both variants the provided protocols are optimal in the sense of the number of used agents — $d + 1$ agents are necessary and sufficient. Visibility has an impact on the time and number of moves and it allows to make the algorithm distributed without a reliance on a coordinator. However, the differences are not significant asymptotically. Both versions require $O(d3^d) = O(n \log n)$ moves and $O(3^d) = O(n)$ time[7]. It is noted that it is a difference of a logarithmic factor from the provided lower bounds.

Agents use $O(\log n)$ bits of memory and have distinct identifiers. A detail associated with the protocol for visibility is that a directed subgraph is constructed where each node requires a small constant number of bits.

Pyramids

Let us begin to close this section with works unaffiliated with Flocchini et al. First, [214] is concerned with cleaning pyramid networks. This specific class of graphs can be unambiguously defined with a single parameter d, which refers to the number of the pyramid's levels. A j-th level is the $2^j \times 2^j$ mesh, where $0 \leq j \leq d$. To preserve the intuition of the visual representation of a pyramid, let 0-th level be the highest, single apex node and d-th mesh be the lowest. Thus, the number of nodes in such a graphs is $n = (4^{d+1} - 1)/3$. Intuitively, additional edges are drawn between meshes of levels differing by one in such a way that each vertex on a higher level corresponds to four vertices below.

The researchers manage to avoid the need for designation of a coordinating agent thanks to the visibility property. Furthermore, the agents are able to clone themselves and start in one of the corners of the lowest level mesh.

Two algorithms are given, one which prioritizes the number of searcher over the time and the other which behaves in a reverse manner. The first one uses $2^{d+1} - 1 = O(\sqrt{n})$ searchers and $3 \cdot 2^d - 1 = O(\sqrt{n})$ steps. The second one $2^2 d = O(n)$ searchers and $4d = O(\log n)$ steps. While in the latter solution the linear

[6]Note that the number of iterations is distinct from the notion of either fractal dimension or an analogous fractal embedded in a higher dimensional space, e.g. Sierpiński tetrahedron. In the relevant context, decontamination of the given example is left as an open question in [178].

[7]It appears that the entry in the survey [194] is incomplete and the discrepancy between the result reported in [178] is left without a commentary.

number of agents is rather unsatisfying, the time of searching is asymptotically optimal.

Stars

Let us dispel the notion that this section is a less interesting investigation of trees by clarifying that [159] researches d-dimensional star graphs. In fact, for this class of graphs $n = |V| = d!$ and $m = |E| = n(d-1)/2$. As with many high dimensional structures, our imagination is prone to failure and recursion is a great help in the design of strategies. The provided algorithm works under the assumption of the node variant augmented by cloning, synchronicity and a sense of whether a neighbouring nodes are clean (i.e. 1-visibility with a reduced set of local knowledge), since no whiteboards and no detection of other agents is used). Note that the authors themselves do not use these terms and define them independently[8].

This work appeared after [158], which will be discussed in the next section, and uses the same concept of a search spanning tree. Based on the properties of searching of trees, the authors estimate that the required number of agents is no more than $1 + \log_3 (n-1)$. The number of steps is at most $3 \lfloor 3(d-1)/2 \rfloor - 2$. The floor corresponds to the height of the search spanning tree or the diameter of the star. Note that since $n = d!$, the researchers are able to claim sublogarithmic time of execution, which compensates for the very generic upper bound on the number of used agents.

Product networks

Given the results for particular classes of graphs one might be interested in finding out an efficient way of decontaminating a graph obtained by combining other graphs for which search strategies are known. To this end, [158] provides algorithms for calculating a strategy for a Cartesian product of graphs, henceforth simplified as product. Let $G_1 \Box G_2$ be the the product of graphs G_1 and G_2. Let us call the final result of applying this operation to a set of graphs a *product network*. Authors point out that this operation is known to preserve, among other properties, regularity, vertex-transitivity and Hamiltonicity. Furthermore, the structures of the graphs given to the operation are preserved as subgraphs of the product network. In a product of $G_1 \Box \ldots \Box G_d$, let component G_i, $i \in \{1, \ldots d\}$ refer to the subgraph isomorphic to G_i. The model is the same as in the researchers' previously discussed work [159] from the section above[9].

The high level nature of the algorithms assumes knowledge of sub-procedures that clean each component G_i that use a known number of agents (here denoted $A(G_i)$) and moves or parallel steps they perform (here denoted $M(G_i)$). Both algorithms utilize the newly introduced concept of a search spanning tree. The first one results in $A(G_1) \prod_{i=2}^{d} |V(G_i)|$ agents cleaning the product network in

[8] The proposed classification is consistent with the one deployed in [197].

[9] [197] does not classify this model as featuring visibility, which appears as an inconsistency.

$\sum_{i=1}^{d} M(G_i)$ moves. In essence, the time used is the sum of the times of the sub-procedures. The second one uses $\prod_{i=1}^{d} |V(G_i)| - 1$ moves, which is simply the number of edges in the search spanning tree. The number of used agents is asymptotically the same, although the given equation is much more complex. Naturally, a proper choice of a homebase and component G_1 gives an upper bound on the monotone connected search number.

Observe that the product operation allows to create some classes of graphs discussed in previous parts of this section. For example, when P_1 and P_2 are paths, then $P_1 \square P_2$ is a mesh (this example is explicitly used in the work itself). Similarly, two cycles produce a torus. Thus, [158] supplements results from [117, 113] with a case including cloning.

2.2.4 Cleaning the unknown

Stumbling in the darkness - experimental approach

Let us consider the case when the graph is *a priori* unknown to searchers. We begin the investigation with the experimental approach in [119] and [120]. The authors considered models with combinations of the properties discussed so far: either locality or 1-visibility or 2-visibility, with or without cloning. Moreover, breaking with the standard approach of the authors in their works on decontamination of specific topologies of graphs, searchers operate in a synchronous fashion and multiple homebases are considered. Hence, the strategy is necessarily not connected, albeit it remains monotone and internal. Since the choice of homebases can significantly alter the performance of their approach, a genetic algorithm is responsible for choosing them. The results, evaluated in terms of the number of agents and the number of synchronous rounds, show an improvement when a greater number of homebases is used.

Surely but slowly

Fully theoretical results on searching graphs of unknown topology were obtained by [30]. In this model, the nodes of the graph are unlabeled and an agent on a node u can distinguish only labels of the ports, denoted by distinct integers from 1 to the degree of u. Distinct identifiers are also assigned to agents. Furthermore, the model is asynchronous and each node is equipped with a whiteboard. At most $O(m \log n)$ bits are stored at every whiteboard, while searchers use at most $O(\log k)$ bits. The provided protocol uses only one searcher more than the optimal monotone connected strategy without knowing the needed number of agents in advance. Instead, a single agent appears at a predestined homebase and is able to call for assistance, or clean the graph by itself if possible.

Note that the authors use monotone connected search number with respect to both the given graph and its single distinguished homebase v_0 (denoted $\mathrm{mcs}(G, v_0)$) node as their baseline of efficiency. However, while the agents are able to compute and ultimately execute the moves based on a monotone connected strategy, their

overall strategy is not monotone, merely connected. As a consequence the number of moves is not restricted and might be exponential.

This, rather disheartening, restrictions are supplemented by the results from [157], where a competitive ratio of $\Theta(n/\log n)$ was established between centralized and distributed monotone connected searching. The results are constructive, the existence of graphs such that no protocol can do better than $c\frac{n}{\log n}\mathrm{mcs}(G, v_0)$, for a certain constant c and a sufficiently large n, is matched by a protocol that achieves this lower bound within a constant factor. Interestingly, the difficult graph in question is a tree with a degree at most 3.

The model is the same as in [30], the assumption of synchronicity is used only to strengthen the lower bound. Searchers use $O(\log n)$ bits of private memory and $O(n)$ bits on whiteboards. The agents make at most $O(k \cdot n \cdot m)$ moves, where k is the number of used agents. Although the original concern of avoiding the exponential number of moves is already addressed by the assumption of monotonicity, it is noted that establishing a competitive ratio for non-monotone connected searching with a polynomial number of moves is an open question.

A recent development was made in [233], where many commonly held properties of the discussed models, including connectivity, were dropped. The motivation refers to Parsons' idea of a dark cave, further denying the searchers any information about its structure. The model is distributed and asynchronous. The authors show that, as long as uniquely labeled agents are able to meet (i.e. solve the rendezvous problem) and coordinate face-to-face, no homebase, whiteboards, node or even port identifiers are necessary to search the graph with the same number of searchers as in the centralised edge search model. No bounds on the time complexity are provided.

Oracles and senses

A measure of difficulty in terms of the size of additional information — *oracle size* or *number of bits of advice*, needed by independent entities to solve the problem efficiently was investigated in [129], in the context of communication tasks by nodes, namely broadcast and wakeup. This measure was applied to distributed graph searching of unknown graphs in [195], not long before [157]. Monotone connected searching model was considered, moreover, using the optimal number of searchers needed in the centralized problem (i.e. $\mathrm{mces}(G)$). It was shown that the desired search can be performed in $O(n^3)$ time given, or provided by an oracle, a labeling of nodes using $O(n \log n)$ bits. Furthermore, this number of bits of advice is optimal, as $\Omega(n \log n)$ bits of advice are shown to necessary for some classes of graphs. The agents themselves are equipped with $O(\log n)$ bits of memory and whiteboards of size $O(\log n)$ are used by the presented protocol only to facilitate face to face communication.

Unknown graphs in a modified edge searching model were considered in [43]. Here the nodes are weighted, and the fugitive may pass through a node guarded by the number of searchers less than its weight. The agents start on an arbitrary node; however, the are equipped with a *sense of direction*. Informally speaking,

it can model formulating a plan where each searcher is equipped with a compass and an ability to measure distance, which allow them to *a priori* agree on the division of the space where the cave is embedded into sectors. Formally speaking, vertices of the graph to be searched are partitioned into t sets $(V_1, \ldots, V_t)$. An edge can only exist between vertices belonging to the same set or two consecutively numbered ones. An agent can recognize whether a port on a currently occupied node $v \in V_i, i \in 1, \ldots, t$ leads to a node in V_i, V_{i+1} or V_{i-1}. Note that this ability can be conceptualized as an oracle that for each ordered pair of vertices (or a port) consistently outputs one of the values from the following set: $\{-1, 0, 1\}$. Thus, the number of bits of advice would be $O(m)$. In this model, the researchers provide an algorithm which uses $3w(G) + 1$ searchers, where $w(G)$ is the maximum over the sums of weights in each V_i. The obtained search strategy is monotone and connected; hence, $\text{mces}(G) \leq 3w(G)+1$. While it is not claimed to be the optimal amount, it is within an additive constant of 2.

Last but not least, [93] describes an algorithm for decontamination of unknown partial grids. Graphs in this class can be imagined as embedded in a two-dimensional Cartesian coordinate system, where nodes are placed on points with integer coordinates and an edge can only exist between nodes within distance one of each other. Thus, with a sufficient density of nodes, such a mesh can approximate any two-dimensional shape. While the agents do not know the graph in advance, they are equipped with a sense of direction in the form of an ability to recognize the four directions on the grid. This is noted to be done thanks to a consistent labeling of ports, although an oracle similar to the one described in the paragraph above can be easily substituted. Some results for the standard model of node searching in known topologies of grid-like graphs were presented in [102]. As authors note, they establish "unobvious proofs that some fairly obvious upper bounds on the pathwidth of some standard grid-like graphs are indeed also lower bounds." Among these, a lower bound on the number of searchers required in a full grid with dimension of equal size equal to $\Omega(\sqrt{n})$. In this sense, the algorithm for connected monotone searching presented in [93] is asymptotically optimal. Furthermore, there exist partial grids such that the competitive ratio between searching in known and unknown graphs (when the homebase is assumed to be chosen by the adversary) is $\Omega(\sqrt{n}/\log n)$.

2.2.5 Black Virus

In this section we consider a decontamination problem of a different nature. Let us consider a much more dangerous fugitive, namely a *black virus* — a harmful agent capable of destroying any searcher which enters a node it occupies. In turn it can be destroyed if it moves to an occupied node. The inspiration is the extension of the concept of harmful environments, commonly referred to as *black holes*. For an overview see [121]. In the case of a black virus the threat is expounded by its ability to clone itself to neighbouring nodes when it destroys an agent, although the original node it occupied becomes clean in the process. Therefore, the agents' strategy consists of surrounding the virus, followed by a suicide attack which in

turn prompts the virus to spread into nodes occupied by agents where the clones perish. Thus, an additional optimization criterion is concerned with minimizing the spread of the virus, i.e. the number of nodes infected in such a way. Since an agent needs to sacrifice itself in order to prompt an instance of the virus to move and give another searcher a chance to destroy it, the parameter is also equal to the number of destroyed agents once all the instances of the virus are removed. Note that once the virus is triggered its location becomes known. Hence, the problem differs considerably from decontamination problems discussed previously, where the fugitive can be conceptualized as always hiding in the last place the agents reach.

The problem was studied first for specific classes of graphs: grids, tori and hypercubes [47]. The scope was later expanded to arbitrary graphs [48]. State of the art knowledge on this topic can also be found in the PhD theses: [46], written by one of the authors of the above works, and [174]. In the latter, agents are allowed to move in parallel, which results in more efficient protocols.

2.2.6 Immunity

This section is concerned with granting the network increased resistance to attacks, namely some conditional *immunity*. Let us distinguish three concepts of granting nodes this grace: two spatial and one temporal.

Local majority

The first spatial concept, researched in [177], is based on a *local majority* rule. In essence, a node is recontaminated only if more than half of its neighbours are contaminated. This model, while weaker than the standard assumption that any contaminated node is capable of spreading its malignant influence, reflects a measure of a fault tolerance in the system, e.g. a requirement of a majority vote to approve of a change. Tight lower and upper bounds on the number of searchers were obtained for the following classes of graphs: d-dimensional toroidal meshes (2^d searchers), trees, and graphs with the maximum vertex degree of three. In this last case, a single agent suffices when there exists a vertex of degree one, or a pair of searchers is necessary otherwise. A more general case of decontaminating b-ary trees of height h can be solved with immunization using $h + 1$ agents when $b \geq 4$ and h agents when $b \geq 3$. Note that, in particular, binary trees were proven to require $\Omega(\log n)$ searchers [11] without immunization. Furthermore, the authors provide protocols achieving these upper bounds and analyze the bounds on the total number of moves made while the agents execute them. Let us remark that toroidal meshes are searched in a different model, one with local communication and synchronicity, in contrast with global communication and asynchronicity for other topologies. Nonetheless, the usual conversion into an asynchronous setting using an additional coordinator agent applies to problems presented in this section.

Threshold

Soon after, a general approach to decontamination with immunity [176] was devised where the immunity is granted upon reaching a threshold of m clean neighbours. This notation was revised in the later works [115, 116], which treat m as a number of adjacent, contaminated edges needed to break the immunity. We follow the latter convention. Thus, when $m = 1$ (or $m = 0$ in the former article), then the problem is equivalent to the commonly studied decontamination variants discussed in the sections above. The three works are jointly concerned with decontamination of meshes, toroidal meshes, hypercubes and trees, although it may be sufficient to familiarize oneself with only the latest work [116], which retraces the old results with the consistent notation. These three works constitute the state of the art of the second branch of decontamination with immunity, namely *threshold* immunity or m-immunity. Note that m is assumed to be the same throughout the graph and the open question of introduction of heterogeneity to the parameter was not followed so far.

Meshes and hypercubes are searched in a quasi-synchronous fashion, i.e. all agents are guaranteed to communicate locally and execute their moves in synchronous rounds. As in the section above, a single additional coordinator agent can facilitate execution of the proposed strategies in asynchronous environments. It is noted that the choice of homebase is trivial in the case of toroidal meshes, due to the symmetry of the environment. However, in ordinary meshes the agents are assumed to start in a corner.

A d-dimensional mesh, denoted D (Z for a toroidal mesh), is unambiguously given in terms of sizes of each dimension, i.e. its number of nodes is $N = \times n_1 \times n_2 \times \ldots \times n_d$ such that $2 \leq n_1 \leq n_2 \leq \ldots \leq n_d$. The number of required searchers are given as a function $\mathcal{A}(G, m)$ of a graph G and m, the *immunity threshold*.

For d-dimensional meshes a protocol requiring $n_1 \times n_2 \times \ldots \times n_{d-m}$ agents is given, where $1 \leq m < d$. A single agent suffices when $m \geq d$. On the contrary, for toroidal d-dimensional mesh Z the provided algorithm requires $2^m \times n_1 \times n_2 \times \ldots \times n_{d-m}$ agents, where $1 \leq m < d$. Furthermore, this number is reduced to 2^{2d-m} when $d \leq m \leq 2d$. Note that while the case of hypercubes is explicitly considered in [115], the last result applies. See [180] for the relation between hypercube and torus. In both cases a linear number of moves (in the terms of N, the size of the mesh) is needed.

Although the intuitive interpretation of these constructive results is that increasing m allows to optimize their guarding positions with respect to the largest dimensions, it is not known whether this is the best that can be done. Indeed, the algorithm given for meshes when $1 \leq m \leq d-1$ is known to be optimal with regards to the number of searchers only when $m = d-1$, while the obtained lower bound is $\mathcal{A}(D, m) \geq n_1 + n_2 \ldots + n_{d-m} - (d-m-1)$. Similarly, for toroidal meshes $\mathcal{A}(Z, m) \geq 2(n_1 + n_2 \ldots + n_{d-m} - (d-m-1))$ is reached by the algorithm only for $m \geq 2d-3 \geq 2d$. Note that the gaps are large since the obtained lower bounds are additive in terms of sizes of dimensions, as opposed to multiplicative expressions of numbers of agents needed by the algorithms. Furthermore, the bounds hold even

in a fully synchronous setting. Nonetheless, the bounds on the number of moves are asymptotically tight.

Furthermore, trees admit an asynchronous protocol, optimal with regard to both number of agents and moves, that accomplishes the task of decontamination. Furthermore, under the assumption of a round taking one unit of time it performs in at most $2n - 3$ rounds. Note that, unlike in the case of meshes, the homebase is given as part of the input.

Temporal

Third, we consider the *temporal* formulation of immunity introduced in [118]. In this model each node cannot be recontaminated after being left unoccupied for t units of time, a given *immunity time*. Note that so far it was always assumed that $t = 0$. No differentiation between immunity times for specific nodes was considered. Naturally, given a sufficient amount of immunity time, any number of agents, even a single one, can decontaminate the whole graph. Thus, given a number of searchers k, a graph's *(monotone) immunity number*, denoted ($\mathrm{mit}_G(k)$) $\mathrm{it}_G(k)$, is the minimum amount of immunity time t required to decontaminate it. Let us further distinguish $\mathrm{mit}_G(k)$ and $\mathrm{it}_G(k)$ by referring them specifically to models from [238] and [74] (the original source of this formulation), respectively. While [118] uses a slightly different notion of temporal immunity, they are nonetheless easily reducible to each other.

Comprehensive results for monotone connected searching of trees are given in [118]. For a tree T, the number of agents required to decontaminate it can be computed in $\Theta(n)$ time in a centralized model, for all possible homebases. The movement of agents is synchronous and, since the strategies are times sensitive, they do not admit a simple conversion into asynchronicity by the familiar introduction of a coordinator. To compute a strategy the authors consider the asynchronous message passing model as described in Santoro's book [210], which, despite being distinguished with a title of "classical", turned out to be a rarity among decontamination problems. Recall, that the model with messages was also used in [11], the introductory work of Section 2.2. $\Theta(n)$ messages are needed between the nodes to mirror the results obtained for the centralized model.

Furthermore, a characterization of a class of forbidden family of subtrees $F(k, t)$, for given t and k, is described. As a consequence, it is shown that there exist trees that require $\Omega(\log_3(n/(t + 1)))$ searchers. Interestingly, $F(k, 0)$ supplements the results in [11] and matches the characterisation of connected graph searching of trees that appeared later in [10], although the connection is not made explicitly in the latter work. Intuitively, as t increases the edges of the trees in the family become subdivided proportionally. Therefore, as the movement along them becomes more time consuming, the agents find themselves unable to omit guarding any vertex present in the family of the same k but lower t.

When the agents' knowledge of the tree is restricted to its height (denoted h) , a protocol that cleans any such tree with $k = \left\lfloor \frac{2h}{t+2} \right\rfloor$ searchers is provided. Here a more familiar face-to-face communication has to suffice, as the nodes cannot ex-

change messages to compute the strategy in advance. The number of agents might not be optimal in the sense of matching the one obtained by the algorithms in the paragraph above. The algorithm exploits the fact that, even though no recontamination is allowed to occur, the agents can repeatedly leave a vertex adjacent to an unoccupied, contaminated edge as long as they return in time to guard it. The assumption of monotonicity was discarded to achieve better results in [74].

The results for 2 dimensional meshes ($m \times n$, $m \geq n$) and toroidal meshes can be found in Daadaa's PhD thesis [238][10].The environment is synchronous, the nodes contain whiteboards and the four directions are consistent for all agents, thanks to a common labeling of ports. Furthermore, the agents are given distinct identifiers and start from a single homebase. Two algorithms that decontaminate a mesh for a given immunity time t are provided. One using $k = \lceil m/(\lceil t/2 \rceil) \rceil$ agents, and the other $k = \lceil (2m-1)/t \rceil$ agents. In the case of toroidal meshes, these numbers are multiplied by 2. Note that no claims of the optimality of these numbers are made. Hence, they can be treated as lower bounds. On the contrary, the results for decontamination with a single agents are tight. Thus, for a mesh M and a torus Z, $\mathrm{mit}_M(1) = 2m - 1$ and $\mathrm{mit}_Z(1) = 2(m-1) + 2(n-1) - 1$, respectively.

The exploration of parameter $\mathrm{it}_G(1)$ for other classes of graphs was continued, with a small alteration, in [74]. In this work, t refers to the uninterrupted time an unoccupied node can have a contaminated neighbour without succumbing to recontamination itself. Observe that if P is a path, a single agents suffices even in a model without immunity, therefore $\mathrm{it}_P(1) = 0$. An instructive example is a cycle of length greater than 4, which can be decontaminated with a single agent when $t = 2$. A simple strategy which makes it possible is to make the agent move in a consistent direction. Since $t = 2$, the agent leaves a growing trail of clean nodes and the cycle is clean when the agent catches up to it. This occurs after at most $2n$ moves. However, the strategy is not monotone.

Upper bounds are also given for: cliques, complete bipartite graphs, meshes, planar, general graphs, spider graphs and trees. Lower bounds are provided for all but the last two classes. Furthermore, lower bounds for monotone strategies are given. Here let us just note that there are cases where non-monotone strategies are asymptotically better and refer a reader interested in details to the table in [74].

We finish with a few non-trivial observations gathered from this work. In some of the cases, e.g. cycles, the time to decontaminate a graph with a single agent can be decreased given a higher t. However, the gain in terms of the asymptotic time is shown to be present only in the case of general trees. Moreover, it can be observed that adding new edges to a graph may decrease its immunity number, i.e. the property is not closed under subgraphs.

[10] As an offhand remark we mention that the bulk of this work is concerned with an approach to decontamination based on cellular automata [73, 72]. A more recent general overview can be also found in [205].

2.2.7 Self-stabilization and its consequences

To the best of my knowledge, there exist only two works on *self-stabilizing* graph searching, [189] and [27], both concerned with only trees. Despite this fact there is an interesting line of research the leaves the realm of distributed graph searching back into its centralized roots.

Although the proofs are concerned with the node searching model, the authors show that their algorithms can also be adapted to edge and mixed searching. The latter work is an improvement on the result of the former, reducing the number of required searchers from being equal to the height of the tree to $1 + \lfloor \log n \rfloor$, in line with $\Theta(\log n)$ searchers required by centralized algorithms. A self-stabilizing model's distinguishing premise is that every node of the graph can communicate only with its neighbours and change its state according to some set of rules. Which nodes get to execute their protocols is decided by an adversary, or in the presented case, strictly speaking, a *distributed unfair daemon*. When the rules followed by the nodes do not permit any changes the global system is said the be in a *stable configuration*. Naturally, in the case of graph searching, the tree should be clean when such a state occurs. In order not to introduce yet another measurement of efficiency, specific to self stabilizing algorithms, let us only remark that the algorithm in [27] executes $O(n \log n)$ moves.

Furthermore, introducing a recontamination from the outside results in resumption of the stabilization process (the algorithm is *non-silent*), thus ensuring that the graph can be cleaned in perpetuity. Under this condition, contamination may be approached as a constant threat, and continuous application of a perpetual strategy ensures that all outbreaks will be handled, eventually. In advance of the discussion of the next work, let us call such a strategy *perpetual*[11].

Perpetual graph searching was considered in another distributed model, namely Look-Compute-Move model, in [28]. The mixed searching model was adapted into explicitly perpetual graph searching, where the next move of a searcher is computed using the position of all searchers (recall the description of the Look-Compute-Move model). This imposes an additional requirement that no two searchers may ever occupy the same node, as the adversary can then make them act in unison. Since the model does not provide any way to break symmetry in such cases, it effectively reduces the agents' capability to reenact the strategy and clean the graph in the future. This particular model also did not get a lot of attention and, to the best of my knowledge, the only obtained results describe solutions for paths and trees [28], and cycles [101, 100]. Nonetheless, the condition that two searchers can never occupy the same node proved to be interesting enough to study the centralized searching without collision model. It appeared in an article of its own, by the same authors as [28], under the name of *exclusive searching* [29], which an attentive reader may recall from Section 1.6.

[11] A reader intrigued by the concept of perpetual strategies might be interested in the following PhD thesis: [188]. The author explores continual process of cleaning a graph with brushes, in a model conceptually related to the edge searching problem and the chip firing game.

Chapter 3

Communication by means of moving information

One of the problems that might arise in a distributed environment is that of sharing information between agents. Since an efficient algorithm solving it could be used as a component of solutions of larger problems, it warrants a closer investigation. The presented approach is based on *delivery problems*, where the task is to move objects, in our case a data packets, from their sources to their destination. In order to move, the information needs to be handled by an agent which can traverse the environment, in our case a network represented by a (weighted) graph.

Bärtschi in his PhD Thesis [15] distinguishes between three optimization criterion for delivery problems:

- energy consumption — the total amount of energy used by all agents;
- delivery time — either of the last package or the sum of the times of delivery;
- resource constraints — no agent can exceed its initially assigned budget.

Despite the assumption of centrality present in many models, the obtained algorithms can be used to schedule movement of mobile agents *a priori*. Furthermore, since the goal of the protocols is the delivery of data, it is natural to assume that agents cannot communicate at a distance. Thus, we return to the world of mostly centralised problems. Nonetheless, some examples of distributed problems will also be discussed. We are going to use the framework of data delivery notation to talk about problems introduced in the subsequent section; hence, some preliminary concepts are presented below.

The problems this work is most concerned with ask whether a delivery is possible under the constraint of a limited available energy. Hence, they fall under the category of resource constraints. Each mobile agent has a limited *battery* (B_i for the i-th agent) which restricts the distance it can traverse (its range). We will call such agents *energy-constrained* (or *power aware*). Note that the amount of energy

given to each agent can be distinct and is given as a part of the input. Nonetheless, when all agents are given the same amount of energy they are said to have *uniform batteries*, or *uniform ranges*. Hence, in a decision version of the problem we ask if there exists a feasible schedule of agents' movements that fulfils a given task such that no agent runs out of energy (i.e. for every agent the used energy does not exceed their given B_i). Since many discussed problems do not have polynomial algorithms when the energy values are exactly sufficient, the notion of a γ-resource-augmented algorithm, which relaxes the energy requirements, is introduced. To give a more precise definition, an algorithm is said to be *γ-resource-augmented* if it finds (in polynomial time) a schedule such that agents use at most $B_i \cdot \gamma$ energy, or recognizes that no such schedule exists.

The environment is represented by a weighted graph, where weights on the edges represent the amount of energy required to traverse them. For the sake of simplicity, we will refer to such an environment as a *network*, with the weights being implicit. Note that there is an easy to overlook detail that might differentiate the discussed models — whether an agent is allowed to finish movement and drop a package on the edge it traverses. If the answer is affirmative, the network is *continuous*, otherwise *discrete*.

3.1 Communication and energy

Let us consider a few problems concerning transfer of data that can be posed to a set of agents in a network environment. The following formulation of three communication problems was stated in [64, 67].

1. Data delivery problem: Given two nodes s (*source*), t (*target*) of a network G, is it possible to transfer the initial packet of information placed at node s to node t?

2. Convergecast problem: Is it possible to transfer the initial information possessed by each agent to a fixed agent?

3. Broadcast problem: Is it possible to transfer the initial information of some agent to all other agents?

We devote a separate section to each of these problems in models where agents relay on batteries. A summary of results in works discussed in the following sections can be found in Tables 3.1, 3.2 and 3.3.

Since the package to be delivered is considered to be information, it can be freely copied and shared between the agents. In fact, these operations are assumed to be trivial, and in the algorithms agents implicitly exchange all available data whenever they meet or simply leave it on every visited node.

The study of the above problems in models with power aware agents was initialized by the research on convergcast in 2012 [3]. The authors note a breadth of inspiration, ranging from the problem of pattern formation by simple, mobile agents [221], the increasing demand for energy efficiency [1], load balancing [7],

graph exploration algorithms where the mobile agent has to resupply its fuel [6] (piecemeal exploration), to studies of the energy efficiency of convergcast when only the messages are sent [162]. Out of these examples we offer further remarks on the last two kinds of works.

First, note that the model where agents are mobile makes these problems distinct from communication in distributed computing environments, where an edge represents a possibility of sending a message between entities. A reader interested in the latter can pick up Santoro's book [210]. Thus, the discussed versions of the problems cannot boast a rich history, despite being based on objectives of well known communication protocols. Nevertheless, vehicle routing better expresses the mobile aspect of the problems and a book on this subject [227] can be found among references in the recent works.

Second, looking at the limited size of battery as a constraint on the range of each agent's movement yields connection with more exploration problems (e.g. [97], where agents are leashed on a rope of limited size) that is more likely to be emphasised in the recent works. A brief mention of further works on tree exploration can be found in Section 3.1.3, concerned with broadcast. Recently, the concept of energy-constraint was applied to near-gathering of agents with limited battery size [16].

3.1.1 Data delivery

Let us begin our investigation with the first of the three problems outlined above. The authors of [50], following he footsteps of [3], consider a model where agents' need to cooperate is forced by their limited energy supplies. Each agent is able to traverse only a limited distance, pick up a package and move it to a different node (keep this assumption in mind). The goal is to deliver data from a given set of source nodes to some terminating node. Note that, according to the classification outlined in the previous chapter, we are interested in the resource constrained version of the data delivery problem, not minimizing the total energy consumption. For an overview of results where total power consumption is considered see Section 3.2.1.

There are several useful observations that simplify this problem. The authors begin with an argument that, without the loss of generality, agents can move sequentially. Moreover, each agent in the sequence moves only once. Hence, they distinguish a variant of the problem where the path from the sink to the source is assumed to be given and fixed. For the case of a single source the problem is NP-Complete in arbitrary graphs [50], even when the batteries are uniform. However, there exist a 2-approximation, in terms of the minimum uniform power (or a distance an agent can travel) available to each agent, and a 3-resource-augmented algorithm. The authors note that combining these two ideas gives a $\min\{3, (1+r)\}$-resource-augmented algorithm, where r is the largest ratio between agent's batteries. For the case of graphs with a fixed path of delivery the problem is also NP-Complete. The result of NP-Completeness of the problem with multiple sources and uniform batteries is shown to follow from [3]. When the graph is a

line and the initial size of batteries can differ, the problem is weakly NP-Hard, which holds even if all input values are integers. In this case, there exists a "quasi-, pseudo-polynomial" time algorithm provided in a separate work [51]. The problem becomes polynomial when the batteries are uniform.

Bärtschi [15] (see also [17]) provides further insight on the problem of data delivery, thanks to the considered version of the *returning* data delivery problem, where agents are required to return to their starting locations after the task is completed. In contrast with the versions discussed above, the returning data delivery problem has a polynomial time algorithm on trees, and a 2-resource-augmented algorithm on general graphs. On the contrary, both versions are NP-Hard on planar graphs, a class that was not investigated before. Furthermore, the thesis provides tight bounds on the existence of *gamma*-resource-augmented algorithms. There exist no $(3-\epsilon)$-resource-augmented algorithm and $(2-\epsilon)$-resource-augmented algorithm for the non-returning and the returning version, respectively. An interesting result that harks back to the preliminary analysis in [50] is that, for a constant number of agents, data delivery can be solved in time $O(n^4)$ by trying out all possible orders in which the agents move. A dynamic programming technique used in this proof allows Bärtschi to drop the assumption of discrete network.

A recent development has been concerned with the fixed path, uniform battery version of the problem. Note that agents without the package can move freely. Hence, it might be advantageous to hand over the package to a single agent multiple times. In turn, the problem can be made easier by explicitly forbidding this behaviour and restricting an agent's ability to carry the package to a single pick-up and drop-off. That is, each agent can carry the package once. [136] provided a 3-approximation (with respect to the batter size) algorithm for this simplified, single hand-over version in discrete networks. On the contrary, [49] describes a $(2-1/2^k)$-approximation algorithm, albeit in continuous networks (2-approximation on directed graphs). In the case of discrete networks, when multiple package hand-overs are allowed and a fixed number of agents is given, the authors prove that the problem is weakly NP-Hard and provide a fully polynomial-time approximation scheme (FPTAS).

A different approach was considered in [67], where agents were given an additional ability to exchange energy when they meet. Thanks to this newfound ability, agents are able to solve data delivery of a single packet in continuous tree and line networks in linear time; however, in the case of arbitrary networks, both directed and undirected, the problem remains NP-Complete. The same work is also concerned with the convergcast problem; hence, it will appear once again in the section below.

Energy exchange was also considered in [8], where the size of agents' batteries, denoted B, was restricted to at most 2 and each edge takes one unit of energy to traverse. Two versions of decision problems which ask for a feasibility of delivery are considered: fixed placement and chosen placement. In the former, agents' initial positions are given as part of the input. In the latter, a subset of nodes (homebases) is given and an agent can be deployed on any such a node of their

choice. Observe that if $B = 1$, then the problem is trivial. Likewise, when the choice of deployment is completely arbitrary (i.e. every node is a homebase), then the agents can simply start on every other other vertex of the path from s to t. Interestingly, when $B = 2$, the problem is NP-Complete in the fixed placement version, while the choice of placement admits a polynomial time solution. The authors note that when generalizing for cases $B \geq 3$ they expect an adaptation of their NP-Completeness proof to suffice. Nonetheless, the status of the choice placement version when $B \geq 3$ is left unclear.

Data delivery models with batteries			
Type	Batteries	Topology	Results
single source		arbitrary	NP-Complete, $\min\{3, (1+r)\}$-resource-augmented[1] [50]
	uniform	arbitrary	NP-Complete, 2-approximation [50]
	uniform	line	polynomial [50]
		line	NP-Complete [50]
		line	weakly NP-Hard, $O(\Delta^2 \cdot n^{1+4\log\Delta})$ time algorithm[2] [51]
fixed path		arbitrary	NP-Complete [50]
fixed path, 1h-o	uniform	discrete arb.	3-approximation [136]
fixed path, 1h-o	uniform	arbitrary	$(2 - 1/2^k)$-approximation [49]
fixed path, 1h-o	uniform	directed arb.	2-approximation [49]
fixed path	uniform	arbitrary	weakly NP-Hard, FPTAS for fixed k [49]
returning		trees	$O(n + k \log k)$ [15, 17]
returning		arbitrary	NP-Complete, 2-resource-augmented[3] [15, 17]
returning		planar	NP-Hard, $(2 - 2/k)$-resource-augmented [15, 17]
single source	energy exch.	tree	linear time [67]
	energy exch.	arbitrary	NP-Complete [67]
fixed placement	low energy	discrete arb.	strongly NP-Hard [8]
chosen placement	low energy	discrete arb.	polynomial [8]
multiple source	uniform	trees	NP-Complete[4] [50]

Table 3.1: Unless stated otherwise, the networks are continuous and there is a single source. Low energy includes energy exchange. 1h-o is a shorthand for a single hand-over.
1. 3-resource-augmented bound is proven tight in [15].
2. When all input values are integers. Δ is the distance between s and t.
3. Tight.
4. Follows from [3].

3.1.2 Convergcast

We begin our exploration of the convergcast problem with the aforementioned [3]. The work considers both centralized and distributed versions of the problem with uniform batteries in continuous networks. In the centralized model, the problem is proved to be strongly NP-Complete in trees, and therefore also in arbitrary networks for which a 2-approximation algorithm is given. Furthermore, the authors mention that their technique proving the NP-Hardness of the problem for trees

extends to the broadcast problem.The work provides linear time algorithms in the case of line networks, for broadcast and convergcast alike. This is the reason why some researchers expressed surprise when the NP-Completness of data delivery on lines [51] was established.

The following assumptions are made in the distributed model: the agents are identical and do not know the network, however they can recognize which ports they use to enter and exit nodes and see each other when they arrive on the same node. An assumption that is not immediately obvious, which however follows from the shape of an optimal strategy in a centralized model, is that the network is truncated in such a way that only the connected part containing all agents is left. Under these conditions, the competitive ratio of 2 of the battery sizes is established for trees. Moreover, this is the best possible result, even for lines. No other optimization criterion is considered.

Recall the energy exchange model [67] mentioned in the section above. Let us stress that the model is concerned with continuous networks, in contrast with the one used in the original research of this thesis in Chapter 5 (based on [66]). The results for convergcast presented there mirror those of the data delivery problem: NP-Completeness in the case of arbitrary networks and a linear algorithm for trees. Observe that, since s, t and positions of agents are given and the problem is centralized, any edge outside of a path leading an agent to either s, t or one of its peers does not need to be visited in an optimal solution. Thus, we can also assume assume the tree to be truncated *a priori*. Furthermore, the authors prove that in any optimal solution each point of the tree is traversed either one time or once in each direction.

Convergcast with batteries			
Type	Batteries	Topology	Results
distributed	uniform	tree	2-competitive algorithm[1] [3]
	uniform	line	linear time [3]
	uniform	tree	strongly NP-Complete [3]
	uniform	arbitrary	2-approximation, NP-Complete [3]
	energy exchange	arbitrary	NP-Complete[67]
	energy exchange	trees	linear time [67]
	energy exchange	discrete trees	linear time [66]

Table 3.2: Networks are continuous and problems are centralised unless stated otherwise.
1. Tight — there is no (2-ϵ)-competitive algorithm on lines, for any $\epsilon > 0$.

3.1.3 Broadcast

We begin this section with an observation that results for broadcast and convergcast are largely the same in [3]. Both problems can be solved in linear time on lines, become NP-Hard even in the case of trees and, in the case of distributed algorithms, do not admit a $(2-\epsilon)$ competitive ratio for any $\epsilon > 0$. However, for the

broadcast problem, the presented algorithms are 4-approximate in the centralized model for arbitrary graphs and 4-competitive in the distributed model for trees. It is not known whether this last result can be improved.

Investigation of the broadcast problem expressed in terms of data delivery in trees was also considered in [65]. The work is concerned with minimizing total energy consumption when all agents start in a single homebase and, in contrast with the NP-Hardness in the model where agents are distributed across the network and have uniform batteries [3], an $O(n \log n)$ time algorithm is provided. The complexity is of the same order regardless whether the homebase and s are distinct or coincide. Furthermore, if the number of agents is unlimited, a linear time is sufficient. Note that the choice of the optimization goal makes using more agents than necessary incur an additional cost. Hence, the number of agents at least equal to the number of leaves suffices to achieve this improvement.

Note that in the case when the source node s coincides with the homebase, the provided algorithm also solves a problem of graph exploration. This immediately results in the problem being NP-Hard for general graphs by reduction to the hamiltonian path problem. Nonetheless, we mention [76, 99, 77] as examples of works where energy-constrained agents are tasked with exploring a tree.

An attempt to work around the NP-Hardness (recall that data delivery on a line was weakly NP-Hard [51]) of the case when the agents are distributed across a discrete tree network was made in [68], thanks to the ability of energy exchange. Note that the assumption of centralization and the possibility of energy exchange in conjunction with a single homebase, as was the case in the previously described work [65], results in uniform and different batteries models being reducible to each other. In this version, the decision problem admits different positions and energy levels of agents while the optimization problem asks the agents to deposit any surplus energy at the source node, therefore maximizing this amount. The presented dynamic programming algorithm solves the problem in $O(n \cdot k^2)$ time, where k is the number of agents. This last work is also a vital prelude to the original research presented in Chapter 5; hence, we delay discussion of some of the algorithm's properties until then.

3.1.4 Gossiping

While the convergcast problem can be thought of as a form of communication from all to one, and the broadcast problem from one to all, their combination, called *gossiping*, asks us to deliver messages from all to all such that in the end all agents possess all available data. The following formulation, attributed to A. Boyd, can be found in [143] (from 1972) and repeated in [145]: *"There are n ladies, and each one of them knows an item of scandal which is not known to any of the others. They communicate by telephone, and whenever two ladies make a call, they pass on to each other, as much scandal as they know at that time. How many calls are needed before all the ladies know all the scandal?"*

This problem was studied in diverse models and contexts: construction of routing tables (chapter 4.2 of [210]), a non-constructive algorithm for radio commu-

Broadcast with batteries			
Type	Batteries	Topology	Results
distributed	uniform	tree	4-competitive algorithm[1] [3]
	uniform	line	linear time [3]
	uniform	tree	strongly NP-Complete [3]
	uniform	arbitrary	4-approximation, NP-Complete [3]
single homebase	unlimited[2]	tree	$O(n \log n)$ time algorithm[3][65]
	energy exchange	discrete tree	$O(n \cdot k^2)$ time algorithm[68]

Table 3.3: Networks are continuous and problems are centralised unless stated otherwise.
1. There is no (2-ϵ)-competitive algorithm on lines for any $\epsilon > 0$.
2. Minimizes total energy. Equivalent to energy exchange since all agents start on the same node.
3. $O(n)$ when the number of agents is at least the same as the number of leaves of the tree.

nication [54], an approximate algorithms for a telephone model [132] and an autonomous approach for wireless mobile ad-hoc networks [78]. A reader interested in the mathematical and graph theory approach to the problem might find the following insightful: old and well established survey [145] and a book from the first decade of this millennium [151]. However, the gossiping problem was not studied in the context of energy aware, mobile agents. This is despite being well known and being a subject of the open questions in [3] and [68], the latter specifically concerned with an energy exchange model. Thus, we state a fourth kind of problems for agents in a network:

4. Gossiping problem: Is it possible to transfer the initial information possessed by each agent to all other agents?

Novel results concerning this problem when agents are allowed to exchange energy will be presented in Chapter 5.

3.2 Heterogeneous problems

Let us begin this outlier section with an unremarkable observation that introduction of a variety in a previously uniform property is a common way of creating a different version of a problem. This section is concerned with such modifications to popular problems; however, not alterations confined exclusively to an environment but rather the entities involved.

Let us classify agent problems as seen through this lens. A problem is said to have a *heterogeneous environment* if a part of its input is a function assigning some different, not necessarily unique, values to a subset of elements of its environment. While the applied name is novel, the concept itself is well established. For example, when the environment is a graph $G = (V, E)$ a function $f : E \mapsto \mathbb{N}$ describes its edge-weighted version. A reader moderately familiar with graph theory is bound

to easily recall their favourite example. Furthermore, an attentive reader will recall that one such problem was mentioned during the discussion of the connected searching of trees [11, 83].

While it is tempting to describe problems as using heterogeneous agents in an analogous fashion, it risks being too broad to capture a meaningful distinction. For example, [94] uses distinct labels in order to ensure a rendezvous of two agents when they don't share a common description of their environment. In the above problem, distinct labels merely allow us to break the symmetry. Hence, we say that the term *heterogeneous agents* applies when agents differ in their abilities. Thus, in our view, only a handful of problems are classified as using heterogeneous agents and will be considered further in this section. Note that heterogeneity of agents is not exclusive with the heterogeneity of their environment, although not every (agent) problem has an immediately obvious way of crafting either modification. Because the breadth of the problems considered in this section we will provide only brief, informal descriptions.

3.2.1 Delivery

Let us begin with a continuation of the previously discussed branch of research in this chapter that could be considered as concerned with heterogeneous agents, namely studying agents operating with limited energy. In Section 3.1, only resource constrained problems were described. However, heterogeneous agents were also explicitly mentioned in Bärtschi's thesis [15] (see also [14, 18]), under the headline of energy-efficient delivery. Hence, in this section the goal is to minimize the sum of spent energy. Note that, unlike in the previous section, networks are always discrete. Since there are multiple reasons why the problem is NP-Hard, mainly algorithms that approximate the energy expenditure are considered.

The agents in this version are weighted, which reflects their rate of energy consumption. Three aspects of the problem are considered. When multiple agents are allowed to handle a packet (*collaboration*) the problem has a polynomial time solution only when a single data packet has to be delivered. Otherwise, when collaboration is forbidden, a 2-approximate algorithm exist. This result made further research significantly easier, as it allows to consider only non-collaborative schedules at a cost of a twofold worse approximation. Even for a single agent, deciding on a route (*planning*) alone is NP-Hard to approximate within a factor less than 367/366. A 1.8-approximation algorithm is provided when agents have *unit capacity*, that is, can carry at most one packet a time. The last aspect, assigning the nodes to packages (*coordination*), is NP-Hard even for unit capacities and when the previously outlined aspects are given and fixed. An exact polynomial solution exists when the agents have the same weights. In essence, when they are not heterogeneous. Given the above results, a general $4r$-approximate algorithm is constructed for delivery with unit capacities, where r is the ratio between the maximum and minimum weights among the agents. Note that r can be arbitrarily large. Finally, we mention a 2-approximate FPT algorithm and a 3.6-approximate algorithm depending exponentially on the number of packets, denoted as m, and

the number of agents, denoted as k, respectively.

The open question whether a polynomial approximation algorithm independent of weights exists, since the algorithms provided by Bärtschi [15] only had this property under the assumption of constant either m or k, was answered affirmatively in [26]. The researchers constructed an 8-approximation FPT algorithm with the running time still exponentially depending on k; however, its complexity allows for a polynomial running time whenever $k = O(\log n)$. Furthermore, its modification achieves a $4k$-approximation in polynomial time. Additionally, when the weights are restricted to only 2 distinct possible values there exists a 36-approximation algorithm. Another obtained result is a general $O(\log^2 n \cdot \log \min\{m, n\} \cdot \log \log n)-$, or simply $\tilde{O}(\log^3 n)$-approximate algorithm.

The two remaining sections of this chapter are concerned with examples of problems from different fields of research.

3.2.2 Heterogeneous rendezvous

We proceed with examples concerning rendezvous in order to better elucidate the distinction from problems relaying on unique labels, such as [222] and the aforementioned [94]. In [88], rendezvous of two agents with different travel times is considered. In this model, the time of traversal of each edge is defined separately for each agent. Hence, one can think of each agent having their own function assigning weights to edges. In the centralized version of the problem, both agents know the graph, their initial locations and both of these functions. On the contrary, in the distributed version agents share the information about the graph and their positions, but each agent can access only its own weight function and private memory. The authors measure the efficiency of their proposed solutions in terms of their relation to the time taken to meet in the optimal solution in the centralized model, denoted T_{opt}, and the number of bits exchanged in communication between the agents prior to the execution of any moves (communication complexity). It is shown that without communication the time required by a distributed solution can be within $O(nT_{opt})$, where n is the number of nodes. Furthermore, a restriction that "one of the agents is always at least as fast as the other one" can ensure that $\Theta(T_{opt})$ time suffices. Additionally, communication complexity $O(n(\log \log(M \cdot n)))$, where M is the greatest weight in both agent's functions, is sufficient to achieve the time of $\Theta(T_{opt})$ even in the case of arbitrary weight functions.

Another example of interplay between heterogeneity and symmetry breaking can be found in [106]. The agents use the information that their speeds are fixed and distinct to achieve rendezvous on a continuous cycle. The authors derive lower bounds and optimize for the speed of rendezvous under different conditions, e.g. allowing agents to communicate through pebbles.

The third example, [56], is concerned with rendezvous of two agents when each one can access only a subgraph, here called *map*, of the whole graph constituting their environment. A similarly formulated problem was also studied earlier in [105] (see also [104]) , where it was shown that in order for rendezvous to be always feasible in the absence of explicit node and agent labels (IDs) one of the

maps must be a subgraph of the other. Furthermore, the model has two more properties, which we describe only informally. First, that the nodes of the full graph can be ordered. Agents do not need to perceive the same order. Second, that the edges have a weight limit that must not be exceeded by the agents. These values are known to agents from their knowledge of their maps. [56] assumes the first result as a given and investigates the four possibilities following from the absence or presence of each of the two subsequent properties, albeit it focuses on asynchronous agents. Let the number of nodes of both maps be denoted as N. In the case when both assumptions hold, the authors contribution is going beyond the adaptation of the algorithm where agents traverse at most $O(N)$ edges provided in [105], which they note would result in a quadratic slowdown. Their technique allows for a solution using $O(N \log N)$ traversals. In both articles, the researchers were unable to provide a general, polynomial in N algorithm when either of the two properties was not given, presenting variety of results of impossibility, analysis restricted only to some classes of graphs or an exponential algorithm [105]. A restriction of access of specific agents to some parts of the graph was considered earlier in the problem of modeling evacuation [42]. While it was not the main feature of the work, the authors claim that the problem restricted to only two types of agents becomes NP-hard.

A very practical approach to rendezvous, this time of more than two heterogeneous agents, was considered in [163]. The entities model closely machines moving on a 2-dimensional plane and a graph models merely a communication network between them. The author allowed for different power levels (with a possibility of energy transfer between agents), sensing range, communication range and moving speed. Furthermore, it mentions collisions as a problem that may be encountered. The provided algorithm's analysis is based on simulations. Hence, it is not strictly a graph problem of the kind analysed in this dissertation; however, it shows some further, sensible directions for choosing properties that could be adapted to theoretical graph environments.

3.2.3 Different speeds

An example of such an adaptation of a model from the realm of heuristic and experiments to the domain of theoretical optimally is boundary, or fence, patrolling. The problem of boundary patrolling, where agents perpetually guard a continuous boundary of a given shape or a line in such a way to minimize the maximum time interval any point can remain unvisited (called *idle time*), was shown to change considerably when agents have distinct maximal speeds [70]. The authors argue that types of algorithms that are sufficient to optimally solve the problem when the speed is the same may no longer be optimal when this is not the case. Their focus is on the number of agents, k, being equal to either 2, 3 or 4, leaving conjectures for the general case of arbitrary k. In particular, the conjecture that a strategy proven to be optimal for $k = 2$ is optimal for any k when the fence is a line was falsified, by counterexample, in [161], despite holding also for $k = 3$. The problem is made more complex in [71], where the agents' ranges of visibility might

also vary.

Next, we briefly mention a very different approach to searching where speeds of agents matter. In [69], the environment is a line and agents are provided with two types of speeds: walking and searching, the latter slower than the former in order to reflect the nature of the arduous but noble task of beachcombing.

A recent work in a field adjacent to data delivery problems discussed in Chapter 3 is [60], concerned with delivery modeled after pony express. This time the speed of agents is subject to differentiation and the goal is optimized delivery time on a line. The authors consider three variants, in a centralized and distributed model. In the case of delivery from one endpoint to the other, both off-line and on-line versions of the problem can be solved in $O(n \log n)$ running time. Half broadcast, i.e. delivery from the center to either endpoint, can be solved off-line in $O(n^2 \log n)$ time, with a competitive ratio $3/2$. For the last variant, broadcast from the center to both endpoints, FPTAS in the off-line case is provided and an on-line algorithm with a competitive ratio $9/5$. The runtime of the approximation schema is $O(n^2 \log n \log 1/\epsilon)$, and this result is obtained using a binary search estimation.

Chapter 4

Heterogeneous graph searching

4.1 Introduction

In view of the fact that our literature review in Chapters 1 and 2 touched many models and subproblems, let us skip the introduction of graph searching as a whole in favour of an informal introduction to the problem of heterogeneous graph searching. Nonetheless, we intentionally may recall certain previously introduced concepts in this chapter. At a price of having potential, small repetitions regarding a few formal terms, we achieve the goal of having full formal context here, in one place, preceding the presentation of results. In particular, in Section 4.2.1, we define agents' available moves and expand the usual definition of recontamination to include the concept of a *unit decontamination*. We note that this chapter is based on [90].

Our basic motivation is based on the idea of decontamination in the field of robotics. To be specific, we seek to completely clear a system of pipes of different shapes contaminated with poisonous gas. In order to do so, we deploy a team of mobile robots such that a specific unit is designed to fit only in a single type of pipe. In short, the searchers are different[1] — each searcher has access only to some part of the graph. This premise is in line with the notion of heterogeneity outlined in 3.2.

Beyond the examples given in the previous section, let us also bring up interdisciplinary uses of graph searching, agents and heterogeneity. For example, Hollinger et al. use graph searching models in their works to guide robots [148, 147]. We refer a practically minded reader to the following survey on the topic [55]. Self-stabilization and Look-Compute-Move models are used to model movement on agents in a polygon environment by [179]. Including heterogeneity to better model real-world scenarios is investigated in [203]. This last work is concerned with traffic flow and models different vehicles as heterogeneous classes.

[1] Let us also mention a few examples of pursuit-evasion games that did not find their way into the review in which some additional device (like a sensor or a trap) is used by the searchers [57, 58, 59, 220].

Our second motivation is an attempt to understand the concept of monotonicity in graph searching. The variant of searching that we introduce has an interesting property: it is possible to construct relatively simple examples of graphs in which multiple recontaminations are required to search the graph with the minimum number of searchers. Moreover, it is interesting that this property holds even for trees. An attentive reader might recall that NP-membership of connected graph searching is an open question [10, 29]. Conversely, the problem is well understood in the class of trees [10], and studying this relatively simple class of graphs seems to offer little help. In this view, exclusive graph searching [29], being non-monotone even for trees, offers an easier case to study (non-)monotonicity in general. Our model shares these benefits.

Last and least, following the footsteps of researchers who provided creative examples of applications of graph searching, we put forward an alternative use of the model of heterogeneous graph searching with a tale. We are given a map that contains rivers and sea routes. Our goal is to hunt an infinitely fast and omniscient, possibly wish granting[2], golden salmon. In order to accomplish this task we are allowed to employ bears to patrol the rivers and sharks to scour the seas. Due to restricted funding, we seek to hire as few animals as possible. One could easily formulate generalizations of such a problem, such as: different wages for each type of entity or an entity that can operate in multiple environments, e.g. a bull shark.

4.1.1 Our work — a short outline

We focus on studying monotonicity and computational complexity of our heterogeneous graph searching problem that we formally define in Section 4.2.1. We start by proving that the problem is not monotone in the class of trees (Section 4.3). Then in Section 4.4 we show that, also in trees, monotone search with heterogeneous searchers is NP-complete. In Section 4.5 we prove that the general, non-monotone, searching problem is NP-hard for trees.

Our investigations suggest that the essence of the problem difficulty is hidden in the properties of the availability areas of the searchers. For example, the problem becomes hard for trees if such areas are allowed to be disconnected. To formally argue that this is the case we give, in Section 4.6, a polynomial-time algorithm that finds an optimal search strategy for heterogeneous searchers in case when each color class induces a connected subtree. This result holds also for the connected version of the heterogeneous graph search problem.

Section 4.2 is concluded with Table 4.1 that points out the complexity and monotonicity differences between the classical and connected edge search with respect to our problem.

[2] [185] does not seem to confirm the last claim.

4.2 Preliminaries

In this work we consider simple edge-labeled graphs $G = (V(G), E(G), c)$, i.e., without loops or multiple edges, where $c\colon E(G) \to \{1, \ldots, z\}$ is a function that assigns labels, called *colors*, to the edges of G. Then, if $c(\{u, v\}) = i$, $\{u, v\} \in E(G)$, then we also say that vertices u and v *have* color i. Note that vertices may have multiple colors, so by $c(v) := \{c(\{u, v\}) : \{u, v\} \in E(G)\}$ we will refer to the set of colors of a vertex $v \in V(G)$.

4.2.1 Problem formulation

We will start by recalling the classical *edge search* problem [199] and then we will formally introduce our adaptation of this problem to the case of heterogeneous searchers.

An (edge) *search strategy* $\mathcal{S}$ for a simple graph $G = (V(G), E(G))$ is a sequence of moves $\mathcal{S} = (m_1, \ldots, m_\ell)$. Each move m_i is one of the following actions:

(M1) placing a searcher on a vertex,

(M2) removing a searcher from a vertex,

(M3) sliding a searcher present on a vertex u along an edge $\{u, v\}$ of G, which results in a searcher ending up on v.

We often write for brevity 'move i' in place of 'move m_i'.

Furthermore, we recursively define for each $i \in \{0, \ldots, \ell\}$ a set $\mathcal{C}_i$ such that $\mathcal{C}_i$, $i > 0$, is the set of edges that are *clean* after the move m_i and $\mathcal{C}_0$ is the set of edges that are clean prior to the first move of $\mathcal{S}$. Initially, we set $\mathcal{C}_0 = \emptyset$. For $i > 0$ we compute $\mathcal{C}_i$ in two steps. In the first step, let $\mathcal{C}'_i = \mathcal{C}_{i-1}$ for moves (M1) and (M2), and let $\mathcal{C}'_i = \mathcal{C}_{i-1} \cup \{\{u, v\}\}$ for a move (M3). In the second step compute $\mathcal{R}_i$ to consists of all edges e in $\mathcal{C}'_i$ such that there *exists* a path P in G such that no vertex of P is occupied by a searcher at the end of move m_i, one endpoint of P belongs to e and the other endpoint of P belong to an edge not in $\mathcal{C}_{i-1}$.[3] We stress out that it is enough that only one such path exists, and in particular, if a contaminated edge is adjacent to a clean edge e, then e becomes contaminated when their common vertex v is not occupied by a searcher. In such case, P consists of the vertex v only. Then, set $\mathcal{C}_i = \mathcal{C}'_i \setminus \mathcal{R}_i$. If $\mathcal{R}_i \neq \emptyset$, then we say that the edges in $\mathcal{R}_i$ become *recontaminated* (or that *recontamination occurs in* $\mathcal{S}$ if it is not important which edges are involved). If l_e is the number of times the edge e becomes recontaminated during a search strategy, then the value $\sum_{e \in E(G)} l_e$ is referred to as the number of *unit recontaminations*. Finally, we define $\mathcal{D}_i = E(G) \setminus \mathcal{C}_i$ to be the set of edges that are *contaminated* at the end of move m_i, $i > 0$, where again $\mathcal{D}_0$ refers to the

[3] We point out that another way of computing the set $\mathcal{C}_i$ is possible. Namely, start again with the same set $\mathcal{C}'_i$. Then, check if the following condition holds: there exists an edge e in $\mathcal{C}'_i$ that is adjacent to an edge not in $\mathcal{C}'_i$ and their common vertex is not occupied by a searcher. In such case, remove e from $\mathcal{C}'_i$. Keep repeating such an edge removal from $\mathcal{C}'_i$ until there is no such edge e. Then, set $\mathcal{C}_i = \mathcal{C}'_i$ and $\mathcal{R}_i = E(G) \setminus \mathcal{C}'_i$.

state prior to the first move. Note that $\mathcal{D}_0 = E(G)$. We require from a search strategy that $\mathcal{C}_\ell = E(G)$.

Denote by $V(m_i)$ the vertices occupied by searchers at the end of move m_i. We write $|\mathcal{S}|$ to denote the number of searchers used by $\mathcal{S}$ understood as the minimum number k such that at most k searchers are present on the graph in each move. Then, the *search number of* G is

$$\mathrm{es}(G) = \min\left\{|\mathcal{S}| \;\middle|\; \mathcal{S} \text{ is a search strategy for } G\right\}.$$

If the graph induced by edges in $\mathcal{C}_i$ is connected for each $i \in \{1, \ldots, \ell\}$, then we say that $\mathcal{S}$ is *connected*. We then recall the *connected search number of* G:

$$\mathrm{cs}(G) = \min\left\{|\mathcal{S}| \;\middle|\; \mathcal{S} \text{ is a connected search strategy for } G\right\}.$$

We now adopt the above classical graph searching definitions to the searching problem we study in this work. For an edge-labeled graph $G = (V(G), E(G), c)$, a search strategy assigns to each of the k searchers used by a search strategy a color: the color of searcher j is denoted by $\tilde{c}(j)$. This is done prior to any move, and the assignment remains fixed for the rest of the strategy. Then again, a search strategy $\mathcal{S}$ is a sequence of moves with the following constraints: in move (M1) that places a searcher j on a vertex v it holds $\tilde{c}(j) \in c(v)$; move (M2) has no additional constraints; in move (M3) that uses a searcher j for sliding along an edge $\{u, v\}$ it holds $\tilde{c}(j) = c(\{u, v\})$. Note that, in other words, the above constraints enforce the strategy to obey the requirement that at any given time a searcher may be present on a vertex of the same color and a searcher may only slide along an edge of the same color. To stress out that a search strategy uses searchers with color assignment $\tilde{c}$, we refer to as a *search $\tilde{c}$-strategy*. We write $\tilde{c}_{\mathcal{S}}(j)$ to refer to the number of searchers with color j in a search strategy $\mathcal{S}$.

Then we introduce the corresponding graph parameters $\mathtt{hs}(G)$ and $\mathtt{hcs}(G)$ called the *heterogeneous search number* and *heterogeneous connected search number* of G, where $\mathtt{hs}(G)$ (respectively $\mathtt{hcs}(G)$) is the minimum integer k such that there exists a (connected) search $\tilde{c}$-strategy for G that uses k searchers.

Whenever we write $\mathrm{es}(G)$ or $\mathrm{cs}(G)$ for an edge-labeled graph $G = (V, E, c)$ we refer to $\mathrm{es}(G')$ and $\mathrm{cs}(G')$, respectively, where $G' = (V, E)$ is isomorphic to G.

We say that a search strategy $\mathcal{S}$ is *monotone* if no recontamination occurs in $\mathcal{S}$. Analogously, for the search numbers given above, we define *monotone*, *connected monotone*, *heterogeneous monotone* and *connected heterogeneous monotone* search numbers denoted by $\mathrm{mes}(G)$, $\mathrm{mces}(G)$, $\mathtt{mhs}(G)$ and $\mathtt{mhcs}(G)$, respectively, to be the minimum number of searchers required by an appropriate monotone search strategy.

The decision versions of the combinatorial problems we study in this work are as follows:

Heterogeneous Graph Searching **Problem** (HGS)

Given an edge-labeled graph $G = (V(G), E(G), c)$ and an integer k, does it hold $\mathtt{hs}(G) \leq k$?

***Heterogeneous Connected Graph Searching* Problem** (HCGS)
Given an edge-labeled graph $G = (V(G), E(G), c)$ and an integer k, does it hold $\texttt{hcs}(G) \leq k$?

In the optimization versions of both problems an edge-labeled graph G is given as an input and the goal is to find the minimum integer k, a labeling $\tilde{c}$ of k searchers and a (connected) search $\tilde{c}$-strategy for G.

	Monotone	Non-monotone	Complexity
ES	arbitrary graphs [199, 202, 190, 186]		P for trees [186], NPC for weighted trees [190], NPC for arbitrary graphs [186]
CES	trees [10, 11]	arbitrary graphs [237]	P for trees [10], NPC for weighted trees [83], NPH for arbitrary graphs [10]
HGS		trees [Theorem 1]	NPH for trees [Theorem 4]

Table 4.1: Monotonicity and complexity summary of our problems in comparison with the classical and connected edge search problems for trees and arbitrary graphs. ES and CES denote edge search and connected edge search, respectively.

4.2.2 Additional notation and remarks

For some nodes v in $V(G)$ we have $|c(v)| > 1$, such connecting nodes we will call *junctions*. Thus a node v is a junction if there exist two edges with different colors incident to v.

We define an *area* in G to be a maximal subgraph H of G such that for every two edges e, f of H, there exists a path P in H connecting an endpoint of e with and endpoint of f such that P contains no junctions. We further extend our notation to denote by $c(H)$ the color of all edges in area H. Note that two areas of the same color may share a junction. Let $Areas(G)$ denote all areas of G. Two areas are said to be *adjacent* if they include the same junction.

Fact 4.2.1. If T is a tree and v is a junction that belongs to some area H in T, then v is a leaf (its degree is one) in H. □

Fact 4.2.2. If T is a tree, then any two different areas in T have at most one common node which is a junction. □

Lemma 4.2.1. Given a tree $T = (V(T), E(T), c)$ and any area H in T, any search $\tilde{c}$-strategy for T uses at least $\text{es}(H)$ searchers of color $c(H)$.

Proof. If there are less than $\text{es}(H)$ searchers of color $c(H)$, then the area H can not be cleaned, as searchers of other colors can only be placed on leafs of H. □

We now use the above lemma to obtain a lower bound for the heterogeneous search number of a graph $G = (V(G), E(G), c : E(G) \to \{1, \ldots, z\})$. Define

$$\beta(G) = \sum_{i=1}^{z} \max\left\{\mathrm{es}(H) \;\middle|\; H \in Areas(G), c(H) = i\right\}.$$

Using Lemma 4.2.1 for each area we obtain the following:

Lemma 4.2.2. For each tree T it holds $\mathtt{hs}(T) \geq \beta(T)$. □

4.3 Lack of monotonicity

Restricting available strategies to monotone ones can lead to increase of heterogeneous search number, even in case of trees. We express this statement in form of the following main theorem of this section:

Theorem 1. There exists a tree T such that $\mathtt{mhs}(T) > \mathtt{hs}(T)$.

In order to prove this theorem we provide an example of a tree $T_l = (V, E, c)$, where $l \geq 3$ is an integer, which cannot be cleaned with $\beta(T_l)$ searchers using a monotone search strategy, but there exists a non-monotone strategy, provided below, which achieves this goal. Our construction is shown in Figure 4.1.

We first define three building blocks needed to obtain T_l, namely subtrees T_1', T_2' and T_l''. We use three colors, i.e., $k \geq 3$. The construction of the tree $T_i', i \in \{1, 2\}$, starts with a root vertex q_i, which has 3 further children connected by edges of color 1. Each child of q_i has 3 children connected by edges of color 2.

For the tree $T_l'', l \geq 3$, take vertices $v_0, \ldots, v_{l+1}$ that form a path with edges $e_x = \{v_x, v_{x+1}\}$, $x \in \{0, \ldots, l\}$. We set $c(e_x) = x \mod 3 + 1$. We attach one pendant edge with color $x \mod 3 + 1$ and one with color $(x - 1) \mod 3 + 1$ to each vertex $v_x, x \in \{1, \ldots, l\}$. Next, we take a path P with four edges in which two internal edges are of color 2 and two remaining edges are of color 3. To finish the construction of T_l'', identify the middle vertex of P, incident to the two edges of color 2, with the vertex v_0 of the previously constructed subgraph.

We link two copies of T_i', $i \in \{1, 2\}$, by identifying two endpoints of the path P with the roots q_1 and q_2 of T_1' and T_2', respectively, obtaining the final tree T_l shown in Fig. 4.1.

Now, we are going to analyze a potential monotone search $\tilde{c}$-strategy $\mathcal{S}$ using $\beta(T_l) = 3$ searchers. Thus, by Lemma 4.2.1, $\mathcal{S}$ uses one searcher of each color. We define a notion of a *step* for $\mathcal{S} = (m_1, \ldots, m_l)$ to refer to some particular moves of this strategy. We distinguish the following steps that will be used in the lemmas below:

1. step $t_i, i \in \{1, 2\}$, equals the minimum index j such that at the end of move m_j all searchers are placed on the vertices of T_i' (informally, this is the first move in which all searchers are present in T_i');

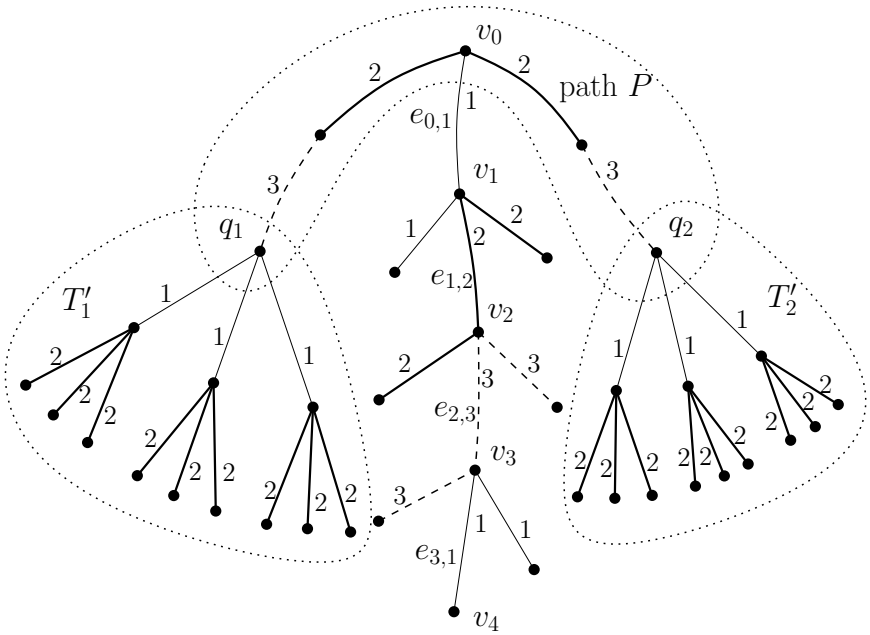


Figure 4.1: The construction of T_3 ($l = 3$) from the trees T_1', T_2' and T_3''. Regular, heavy and dashed edges have labels $1, 2$ and 3, respectively.

2. step $t_i', i \in \{1, 2\}$, is the maximum index j such that at the end of move m_j all searchers are placed on the vertices of T_i' (informally, this is the last move in which all searchers are present in T_i');

3. steps t_3, t_3' are, respectively, the minimum and maximum indices j such that at the end of move m_j all searchers are placed on the vertices in $V(P) \cup V(T_l'')$.

We skip a simple proof that all above steps are well defined, i.e., for any search strategy using 3 searchers for T each of the steps t_i, t_i', $i \in \{1, 2, 3\}$, must occur (for trees T_1' and T_2' this immediately follows from $\mathrm{es}(T_i') = 3$ for $i \in \{1, 2\}$).

Lemma 4.3.1. For each monotone $\tilde{c}$-search strategy $\mathcal{S}$ for T_3 it holds: $t_1 \leq t_1' < t_3 \leq t_3' < t_2 \leq t_2'$ or $t_2 \leq t_2' < t_3 \leq t_3' < t_1 \leq t_1'$.

Proof. Intuitively, we prove the lemma using the following argument: in the process of cleaning T_i', $i \in \{1, 2\}$, all three searchers are required for some steps, and therefore a monotone strategy could not have partially cleaned T_{3-i}' or T_3'' prior to this point.

The arguments used to prove this lemma do not use colors, so atomic statements about search strategies for subgraphs can be analyzed using simple and well known results for edge search model. Furthermore, due to the symmetry of T, it is enough to analyze only the case when $t_1 < t_2$. Note that $t_i \leq t_i', i \in \{1, 2, 3\}$, follows

directly from the definition. The vertices q_i, $i \in \{1,2\}$, have to be guarded at some point between move t_i and move t'_i because $\mathrm{es}(T'_i) = 3$. Because each step t_j, $j \in \{1,2,3\}$, uses all searchers, it cannot be performed if a searcher preventing recontamination is required to stay outside of subtree related to the respective step. The subtrees T'_1 and T'_2 contain no common vertices, so $t_1 < t_2$ implies $t_2 > t'_1$, as stated in the lemma.

Suppose for a contradiction that $t_3 < t_i$ for each $i \in \{1,2\}$. In move t_3, since neither of moves t'_i has occurred, both subtrees T'_1, T'_2 contain contaminated edges. Moreover, some of the contaminated edges are incident to vertices q_i. Thus, any edge of T''_l that is clean becomes recontaminated in the step $\min\{t_1, t_2\}$. Therefore, $t_1 < t_3$ as required.

Now we prove that $t'_1 < t_3$. Suppose for a contradiction that $t_1 < t_3 < t'_1$. Consider the move of index t'_1. By $t_3 < t'_1$, T''_l contains clean edges. By $t'_1 < t_2$, q_2 is incident to contaminated edges in T'_2. Thus, there is a searcher outside of T'_1 which prevents recontamination of clean edges in T''_l. Contradiction with the definition of t'_1.

In move t_2 there are no spare searchers left to guard any contaminated area outside T'_2 which bypasses q_2 and could threaten recontamination of T'_1, so all edges, including the ones in T''_l, between those two trees should have been clean already. Therefore step t'_3 has to have already occurred, which allows us to conclude $t'_3 < t_2$. □

Due to the symmetry of T_l, we consider further only the case $t_1 \leq t'_1 < t_3 \leq t'_3 < t_2 \leq t'_2$.

Lemma 4.3.2. During each move of index $t \in [t'_1, t_2]$ there is a searcher on a vertex of P.

Proof. By $t \geq t'_1$, q_1 is incident to some clean edges of T'_1. By $t \leq t_2$, q_2 is incident to some contaminated edges from T'_2. Hence there has to be a searcher on q_1, q_2 or a vertex of the path P between them to prevent recontamination. □

Let $f_i, i \in \{1, \ldots, l-1\}$, be the index of a move such that one of the edges incident to v_i is clean, one of the edges incident to v_i is being cleaned and and all other edges incident to v_i are contaminated.

Notice that $\mathrm{es}(T''_l) = 2$, and therefore an arbitrary search strategy $\mathcal{S}'$ using two searchers to clean a subtree without colors that is isomorphic to T''_l follows one of these patterns: either the first searcher is placed, in some move of $\mathcal{S}'$, on v_1 and throughout the search strategy it moves from v_1 to v_{l-1} or the first searcher starts at v_{l-1} and moves from v_{l-1} to v_1 while $\mathcal{S}'$ proceeds. If for each $i \in \{1, \ldots, l-1\}$ the edge $\{v_{i-1}, v_i\}$ becomes clean prior to the edge $\{v_i, v_{i+1}\}$ — we say that such $\mathcal{S}'$ *cleans* T''_l *from* v_1 *to* v_{l-1} and if the edge $\{v_{i-1}, v_i\}$ becomes clean after $\{v_i, v_{i+1}\}$ — we say that such $\mathcal{S}$ *cleans* T''_l *from* v_{l-1} *to* v_1.

Lemma 4.3.3. Each move of index $f_i, i \in \{1, \ldots, l-1\}$, is well defined. Either $f_1 < f_2 < \ldots < f_{l-2} < f_{l-1}$ or $f_{l-1} < f_{l-2} < \ldots < f_2 < f_1$.

Proof. Consider a move of index f which belongs to $[t_3, t'_3]$ in a search strategy $\mathcal{S}$. By Lemma 4.3.1 and Lemma 4.3.2, a searcher is present on a vertex of P in the move of index f. Hence, only two searchers can be in T''_l in the move f, so $\mathcal{S}$ cleans T''_l from v_1 to v_{l-1} or cleans T''_l from v_{l-1} to v_1. Note that during an execution of such a strategy there occur moves which satisfy the definition of f_i, and therefore there exists well defined f_i. When $\mathcal{S}$ cleans T''_l from v_0 to v_l, then $f_1 < f_2 < \ldots < f_{l-2} < f_{l-1}$ is satisfied and when $\mathcal{S}$ cleans T''_l from v_l to v_0, then $f_{l-1} < f_{l-2} < \ldots < f_2 < f_1$ is satisfied. □

Lemma 4.3.4. There exists no monotone search $\tilde{c}$-strategy that uses 3 searchers to clean T_l when $l \geq 7$.

Proof. We use the following intuition in the proof: whenever a search strategy tries to clean the path composed of the vertices $v_0, \ldots, v_{l+1}$, together with the corresponding incident edges, then it periodically needs searchers of all three colors on this path. While doing this, different vertices of the path P need to be guarded. More precisely, when the search moves along the former path, it needs to move along P as well. Due to the fact that l is large enough, the path P is not long enough to avoid recontamination.

The vertex $v_i, i \in \{1, \ldots, l-1\}$, is incident to edges of colors $i \mod 3 + 1$ and $(i-1) \mod 3 + 1$, and therefore each move f_i uses both searchers of colors $i \mod 3 + 1$ and $(i-1) \mod 3 + 1$. By Lemma 4.3.2, the third searcher, which is of color $(i-2) \mod 3 + 1$, stays on P.

Consider a sequence $f_6 < f_5 < \ldots < f_2 < f_1$. Note that it implies that T''_3 is cleaned from v_{l-1} to v_1. Let us show that it is impossible to place a searcher on the vertices of P such that no recontamination occurs in each $f_i, i \in \{1, \ldots, 6\}$.

Consider the move of index f_6, where searchers of colors 1 and 3 are in T''_l and 2 is on P. Before move f_6 an edge incident to v_6 is clean (by definition of f_6). No edge incident to v_1 is clean and, by Lemma 4.3.1, T'_j has a clean edge, $j \in \{1, 2\}$. In order to prevent recontamination of T'_j, the searcher is present on P, particularly on a vertex of the path from q_j to v_0. It cannot be the vertex q_j, because $2 \notin c(q_j)$, so the edge of color 3 incident to q_j is clean, and the searcher is on one of the remaining two vertices. Consider the move of index f_5, in which the searcher of color 1 is on a vertex v of P. The vertex between q_j and v_0 cannot be occupied, due to its colors, and occupying q_j would cause recontamination — only the vertex v_0 is available, $v = v_0$. Consider the move of index f_4. The vertex v_0 cannot be occupied, due to its colors. The edge e_0 cannot be clean before e_4 is clean, because T''_l is cleaned from v_{l-1} to v_1. Therefore, the searcher on v_0 cannot be moved towards q_2. Monotone strategy fails.

The argument is analogical for a sequence $f_1 < f_2 < \ldots < f_5 < f_6$. By Lemma 4.3.3, T''_l is cleaned either from v_1 to v_{l-1} or the other way, which implies that considering the two above cases completes the proof. □

Lemma 4.3.5. There exists a non-monotone $\tilde{c}$-strategy $\mathcal{S}$ that cleans T_l using three searchers for each $l \geq 3$.

Proof. The strategy we describe will use one searcher for each of the three colors. The strategy first cleans the subtree T_1' (we skip an easy description how this can be done) and finishes by cleaning the path connecting q_0 with v_0. Denote the vertex on the path from v_0 to q_2 as v.

Now we describe how the strategy cleans T_l'' from v_1 to v_{l-1}. For each $i \in \{1, \ldots, l\}$, the vertex v_i is incident to edges of colors $i \mod 3 + 1$ and $(i-1) \mod 3 + 1$ therefore each move f_i uses both searchers of colors $i \mod 3 + 1$ and $(i-1) \mod 3 + 1$. By Lemma 4.3.2, the third searcher which is of color $(i-2) \mod 3+1$, stays on P. Informally, while progressing along T_l'', the strategy makes recontaminations within the path P.

We will define *j-progress*, $j \in \{1, \ldots, l-1\}$, as a sequence of consecutive moves which clean edges of colors in $c(v_j)$ in T_l'' and contains the move of index f_j. Similarly, we introduce $i\text{-}reconfig(u)$, $i \in \{1, 2, 3\}$, as a minimal sequence of consecutive moves, such that there is a searcher on some vertex u of P in the first move of $i\text{-}reconfig(u)$ and the searcher of color i is present on P in the last move of $i\text{-}reconfig(u)$. Let u_b be the occupied vertex of P after the last move of b-th $i\text{-}reconfig(u)$ in $\mathcal{S}$. Additionally let $u_0 = v_0$. Clean T_l'' by iterating for each $j \in \{1, \ldots, l-1\}$ (in this order) the following: $a\text{-}reconfig(u_{j-1})$, followed by *j-progress*, where $a = (j-2) \mod 3 + 1$.

Because determining moves in *j-progress* is straightforward, as they correspond to those in monotone $\tilde{c}$-strategy when $f_1 < f_2 < \ldots < f_{l-2} < f_{l-1}$, we focus on describing $i\text{-}reconfig(u)$ for each $i \in \{1, 2, 3\}$. $1\text{-}reconfig(u_0)$ consists of a sliding move from v_0 to v and a move which places the searcher of color 3 on $u_1 = v$. $2\text{-}reconfig(u_1)$ consists of a sliding move from v to v_0, which causes recontamination, and a move which places the searcher of color 1 on $u_2 = v_0$. $3\text{-}reconfig(u_2)$ does not contain any sliding moves and places the searcher of color 2 on $u_3 = v_0$. Because $a = (j-2) \mod 3 + 1$ and $u_{j-1} = u_{j+2}$ $a\text{-}reconfig(u_j)$ is identical to $a + 3\text{-}reconfig(u_{j+3})$, thus we can describe a strategy which cleans T_l'' for any given l.

When T_l'' is clean, the vertex v_0 is connected to a clean edge and the remaining edges of path P can be searched without further recontaminations. The strategy cleans subtree T_2' in the same way as a monotone one.

Note that the proposed strategy requires new recontamination whenever a sequence of f_i of length 3 repeats itself. Thus, this $\tilde{c}$-strategy cleaning T_l has $\Omega(l)$ unit recontaminations. Note that the size of the tree T_l is $\Theta(l)$. □

Lemma 4.3.5 provides a non-monotone search $\tilde{c}$-strategy which succeeds with fewer searcher than it is possible for a monotone one, as shown in lemma 4.3.4, which proves Theorem 1.

Theorem 2. There exist trees such that each search $\tilde{c}$-strategy that uses the minimum number of searchers has $\Omega(n^2)$ unit recontaminations.

Proof. As a proof we use a tree H_l obtained through a modification of the tree T_l. In order to construct H_l, we replace each edge on the path P with a path P_m containing m vertices, where each edge between them is in the same color as

the replaced edge in T_l. Clearly $\mathtt{hs}(H_l) = \mathtt{hs}(T_l)$. Note that we can adjust the number of vertices in T_l'' and P_m of H_l independently of each other. While the total number of vertices is $n = \Theta(m + l)$, we take $m = \Theta(n)$, $l = \Theta(n)$ in H_l.

In order to clean H_l, we employ the strategy provided in theorem 4.3.5 adjusted in such a way, that any sliding moves performed on edges of P are replaced by $O(m)$ sliding moves on the corresponding paths of P_m. As shown previously, the number of times an edge of P in T_l, or path P_m in H_l, which contains $\Theta(m)$ elements, has to be recontaminated depends linearly on size of T_l''. In the later case the $\tilde{c}$-strategy cleaning H_l has $\Omega(ml) = \Omega(n^2)$ unit recontaminations . □

4.4 NP-hardness for trees

We show that the decision problem HGS is NP-complete for trees if we restrict available strategies to monotone ones. Formally, we prove that the following problem is NP-complete:

Monotone Heterogeneous Graph Searching **Problem** (MHGS)
Given an edge-labeled graph $G = (V(G), E(G), c)$ and an integer k, does it hold $\mathtt{mhs}(G) \leq k$?

Thus, the rest of this section is devoted to a proof of the following theorem.

Theorem 3. The problem MHGS is NP-complete in the class of trees.

In order to prove the theorem, we conduct a polynomial-time reduction from Boolean Satisfiability Problem where each clause is limited to at most three literals (3-SAT). The input to 3-SAT consists of n *variables* $x_1, \ldots, x_n$ and a Boolean formula $C = C_1 \wedge C_2 \wedge \cdots \wedge C_m$, with each *clause* of the form $C_i = (l_{i,1} \vee l_{i,2} \vee l_{i,3})$, where the *literal* $l_{i,j}$ is a variable x_p or its negation, $\overline{x_p}$, $p \in \{1, \ldots, n\}$. The answer to decision problem is `YES` if and only if there exist an assignment of Boolean values to the variables $x_1, \ldots, x_n$ such that the formula C is satisfied.

Given an input to 3-SAT, we construct a tree T_{SAT} which can be searched monotonously by the specified number of searchers if and only if the answer to 3-SAT is `YES`. We start by introducing the colors and, informally speaking, we associate them with respective parts of the input:

- color $\mathtt{V}_p$, $p \in \{1, \ldots, n\}$, represents the variable x_p,
- color $\mathtt{F}_p$ (respectively $\mathtt{T}_p$), $p \in \{1, \ldots, n\}$, is used to express the fact that to x_p may be assigned the Boolean value *false* (*true*, respectively),
- color $\mathtt{C}_d$, $d \in \{1, \ldots, m\}$, is associated with the clause C_d.

We will also use an additional color to which we refer as `R`.

We denote the set of all above colors by $\mathcal{Q}$. Note that $|\mathcal{Q}| = 3n + m + 1$. In our reduction we set $k = |\mathcal{Q}| + 1 + m$ to be the number of searchers.

The construction of the tree starts with a path P of color `R` consisting of $l = 4n + 3m + 4 + 1$ vertices $v_i, i \in \{1, 2, \ldots, l\}$. We add 2 pendant edges of

color R to both v_1 and v_l. Define a subgraph H_z (see Figure 4.2(a)) for each color $z \in \mathcal{Q} \setminus \{\mathtt{R}\}$: take a star of color R with three edges, attach an edge e of color z to a leaf of the star and then attach an edge e' of color R to e, so that the degree of each endpoint of e is two. For each $z \in \mathcal{Q} \setminus (\{\mathtt{R}\} \cup \{\mathtt{C}_1, \ldots, \mathtt{C}_m\})$ take a subgraph H_z and

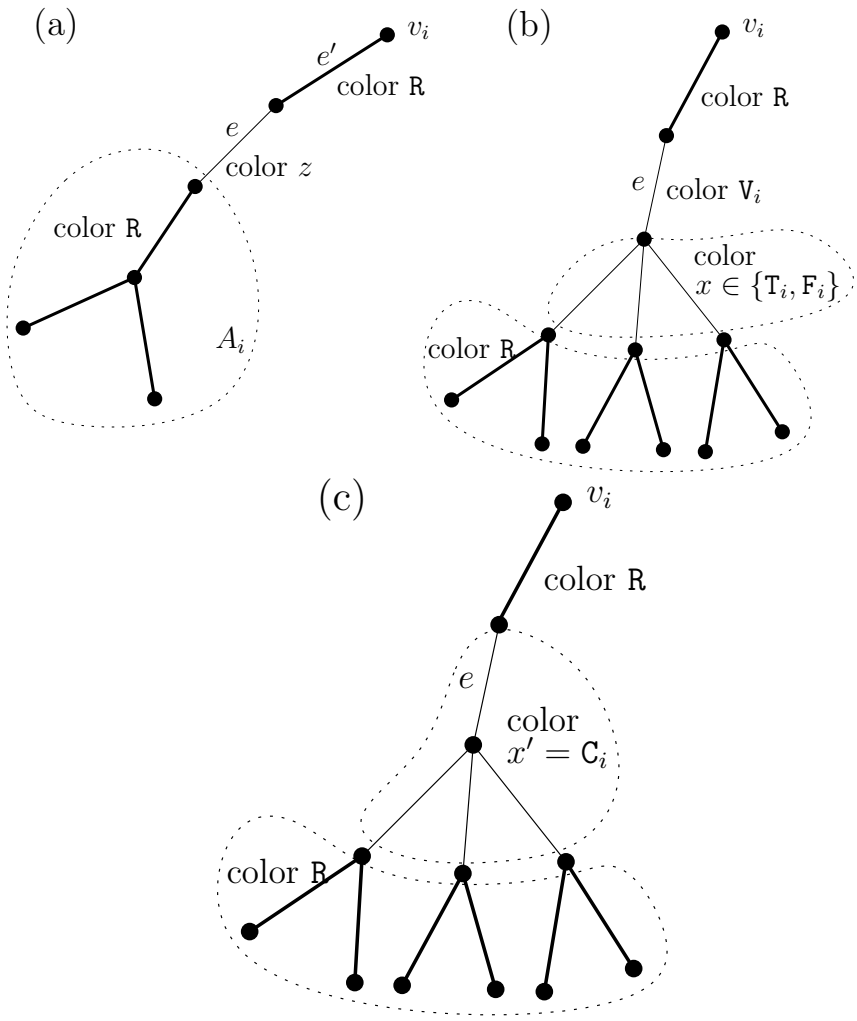


Figure 4.2: Construction of T: (a) the subgraph H_z; (b) the subgraph L_x; (c) the subgraph $L'_{x'}$

join it with P in such a way that the endpoint of e' of degree one in H_z is identified with a different vertex in $\{v_2, \ldots, v_a\}$, $a = 3n + 2m + 1$. For each $z \in \{\mathtt{C}_1, \ldots, \mathtt{C}_m\}$ take two copies of H_z and identify each endpoint of e' of degree one in H_z with a different vertex in $\{v_2, \ldots, v_a\}$, which has no endpoint of e' attached to it yet. The above attachments of the subgraphs H_z are performed in such a way that the

degree of v_i is three for each $i \in \{2, \ldots, a\}$ (see Figure 4.3). We note that, except for the requirement that no two subgraphs H_z are attached to the same v_i, there is no restriction as to which H_z is attached to which v_i. The star of color R in the subgraph H_z attached to the vertex v_i is denoted by A_i.

For each color x in $X = \{\mathtt{T}_i, \mathtt{F}_i \mid i \in \{1, \ldots, n\}\}$ we define a subtree L_x (see Figure 4.2(b)). We start with a root having a single child and an edge of color R between them. Then we add an edge e of color $\mathtt{V}_i$ to this child, where i is selected so that it matches x which is either $\mathtt{T}_i$ or $\mathtt{F}_i$. The leaf of e has three further children attached by edges of color x. We finish by attaching 2 edges of color R to each of the three previous children. For each color $x' \in X' = \{\mathtt{C}_1, \ldots, \mathtt{C}_m\}$ construct a subtree $L'_{x'}$ (see Figure 4.2(c)) in the same shape but colored in a different way. The edges of color different than R in the construction of L_x are replaced by edges of color x'. We draw attention to the fact that $L'_{x'}$ contains an area of color x' that is a star with four edges. We attach to the path P five copies of subtree L_x for each $x \in X$ and five copies of $L'_{x'}$ for each $x' \in X'$ by unifying their roots with the vertex v_{a+1} of P (see Figure 4.3).

We attach five further copies of $L'_{x'}$ for each $x' \in X'$ by unifying their roots with the vertex v_{l-1} of P.

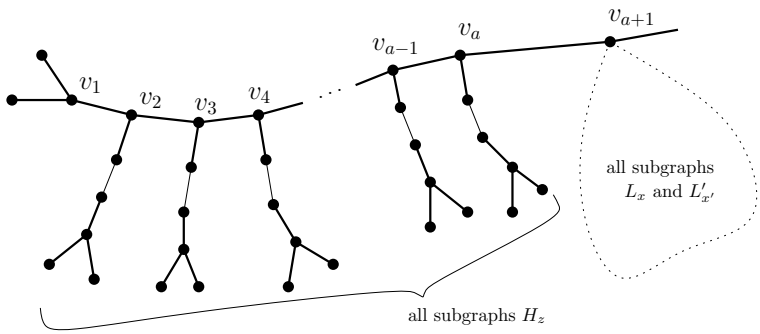


Figure 4.3: Construction of T: attachment of subgraphs H_z and the subgraphs L_x and $L'_{x'}$ to the path P

For each variable x_p we construct two subtrees, S_p and S_{-p}, in the following fashion (see Figure 4.4(a)): take a star of color R with three edges and attach an endpoint of a path with four edges to a leaf in this star of color R; the consecutive colors of the path, starting from the endpoint at the star of color R are: $\mathtt{V}_p$, R, $\mathtt{F}_p$, R in S_{-p} and $\mathtt{V}_p$, R, $\mathtt{T}_p$, R in S_p. For each subtree S_p and S_{-p}, $p \in \{1, \ldots, n\}$ attach the endpoint of its path of degree one to v_{a+1+p} . The star of color R in S_p attached to v_i is denoted by A_i and the one in S_{-p} by A_{-i}.

For each clause $C_d, d \in \{1, \ldots, m\}$ we attach three subtrees $L_{d,j}, j \in \{1, 2, 3\}$, to the vertex v_{b+d}, where $b = 4n+2m+3$, one for each literal $l_{d,j}$ (see Figure 4.4(b)). Note that the maximal value of $b+d$ is $l-2$. We construct $L_{d,j}$ by taking an edge e of color R and adding three edges to its endpoint: two of color $\mathtt{C}_i$, and one either of color $\mathtt{T}_p$ if $l_{d,j} = x_p$ or of color $\mathtt{F}_p$ if $l_{d,j} = \overline{x_p}$. Add two children by the edge of

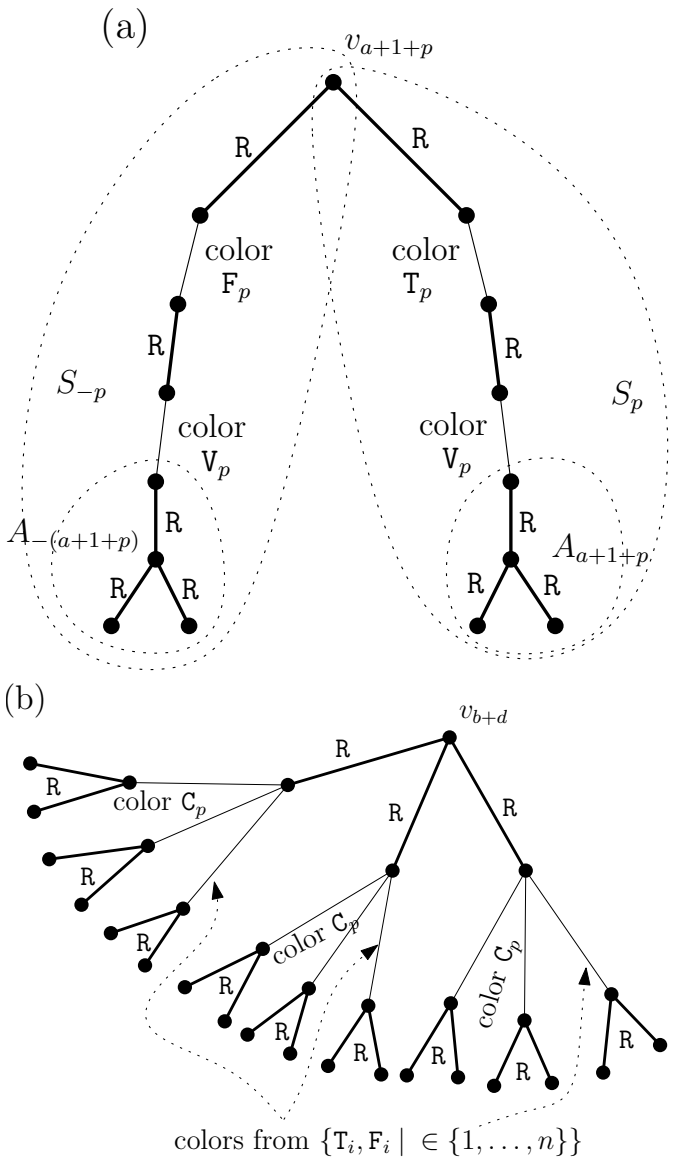


Figure 4.4: Construction of T: (a) the variable component constructed from S_{-p} and S_p; (b) the clause component that corresponds to C_d.

color R to each of these three edges. Then attach the endpoint of degree one of the edge e in $L_{d,j}$ to v_{b+d}. We attach a single edge of color R to v_b. The tree obtained through this construction will be denoted by T_{SAT}.

The area of color R which contains the path P is denoted by A_0. Notice that all areas A_i of color R have search number two, $\mathrm{es}(A_i) = 2$. For a search strategy for T_{SAT}, we denote the index of the first move in which all searchers of color R are in the area A_i as *step* t_i and the index of the last such move as *step* t'_i, $i \in I = \{2, \ldots, a\} \cup \{a+2, \ldots, b-1\} \cup \{-(b-1), \ldots, -(a+2)\} \cup \{0\}$. Let $R = \{a+2, \ldots, b-1\} \cup \{-(b-1), \ldots, -(a+2)\}$ and $L = I \setminus \{R \cup \{0\}\} = \{2, \ldots, a\}$ be the two sets which cover all indices of areas $A_a : a \in I \setminus \{0\}$. Note that by definition $a+1 \notin L$ and $a+1 \notin R$, and the path from v_1 to v_{a+1} contains no vertex $v_j, j \in R$. Similarly, the path from v_{a+1} to v_l contains no vertex $v_i, i \in L$. Informally, we divide the indices in $I \setminus \{0\}$ into two sets: L to the left of v_{a+1} and R to the right.

Lemma 4.4.1 (Color assignment)**.** A search $\tilde{c}$-strategy using $k = 3n + 2m + 2$ searchers has to color them in the following fashion: one searcher for each color in $\{\mathtt{T}_p, \mathtt{F}_p, \mathtt{V}_p \mid p \in \{1, \ldots, n\}\}$ and two searchers for each color in $\{\mathtt{R}\} \cup \{\mathtt{C}_1, \ldots, \mathtt{C}_m\}$.

Proof. We first compute the lower bound $\beta(T_{\mathrm{SAT}})$. By Lemma 4.2.1, at least $3n$ searchers take colors $\mathtt{F}_p$, $\mathtt{T}_p$ and $\mathtt{V}_p, p \in \{1, \ldots, n\}$. Recall that $L'_{x'}$ contains as a subgraph an area T' of color $x' \in \{\mathtt{C}_1, \ldots, \mathtt{C}_m\}$ that is a star with three edges and hence $\mathrm{es}(T') = 2$. Since there are m such subtrees $L'_{x'}$, $2m$ searchers receive colors $\mathtt{C}_1, \ldots, \mathtt{C}_m$. The last two searchers have to be of color R in order to clean areas $A_i, i \in I$. Thus, we have shown that $\beta(T_{\mathrm{SAT}}) \geq 3n + 2m + 2$ and this lower bound is met by the assignment of colors to searchers, as indicated in the lemma. Using Lemma 4.2.2 we complete the proof. □

Lemma 4.4.2. Let $x_1, \ldots, x_n$ and a Boolean formula $C = C_1 \wedge C_2 \ldots \wedge C_m$ be an input to 3-SAT. If the answer to 3-SAT is YES, then there exists a search $\tilde{c}$-strategy using $2 + 3n + 2m$ searchers for T_{SAT}.

Proof. We first note the main point as to how a Boolean assignment provides the corresponding search strategy. Whether a variable is true or false, this dictates which searcher, either of color $\mathtt{F}_p$ or $\mathtt{T}_p$, is placed in the corresponding variable component. The vertices occupied by these searchers form a separator that disconnects A_0 from areas A_i, $i \in R$. The strategy cleans first the latter areas that are protected from recontamination. As a result, all A_i, $i \in R$, become clean. Then, A_0 is cleaned with the clause components along the way: here the fact that the initial Boolean assignment was satisfied ensures that searchers of appropriate colors are available to clean the subsequent clause components.

Suppose that a Boolean assignment to the variables satisfies C. The strategy is described as a sequence of instructions.

1. We start by placing a searcher of color dictated by the Boolean assignment in each variable component. For each S_p (respectively S_{-p}), place a searcher of color $\mathtt{T}_p$ (respectively $\mathtt{F}_p$) on the vertex that is incident to the edge of color $\mathtt{T}_p$ and does not belong to A_0 if x_p is *false* (respectively *true*).

2. Then, clean A_{a+1+p} (respectively $A_{-(a+1+p)}$) and then the edges in S_p (respectively S_{-p}) that connect this star of color $\mathtt{R}$ to the vertex guarded by the searcher of color $\mathtt{T}_p$ (respectively $\mathtt{F}_p$). Note that this cleaning uses two searchers of color $\mathtt{R}$ and the searcher of color $\mathtt{V}_p$. Then, place the searcher of color $\mathtt{V}_p$ on the vertex that belongs to the edge of color $\mathtt{V}_p$ in S_{-p} (respectively S_p) and area $A_{-(a+1+p)}$ (respectively A_{a+1+p}). Clean $A_{-(a+1+p)}$ (respectively A_{a+1+p}).

   By repeating the above for each index p, we in particular obtain that all areas A_i, $i \in R$ are clean. Note that if $x_p = false$ (respectively $x_p = true$), then the searcher of color $\mathtt{T}_p$ (respectively $\mathtt{F}_p$) stays in S_p (respectively S_{-p}) and the searcher of color $\mathtt{F}_p$ (respectively $\mathtt{T}_p$) is available. Let X denote the colors of available searchers among those in colors $\mathtt{F}_p$ and $\mathtt{T}_p$.

3. Then we start cleaning A_0 from the vertex v_l and move towards v_b. Clean copies of $L''_{\mathtt{C}_d}, d \in \{1, \ldots, m\}$ attached to v_{l-1}. Consider each approached vertex $v_h, h \in \{b+1, b+2, \ldots, l-2\}$, and its three subtrees $L_{d,i}, i \in \{1,2,3\}, d \in \{1, \ldots, m\}$ separately. We denote the color different than $\mathtt{R}$ and $\mathtt{C}_d$ in $L_{d,i}$ as $x_{d,i}$. Because C is satisfied, at least one of the literals in each clause is true and for each $d \in \{1, \ldots, m\}$ there always exists $L_{d,i}$ such that $x_{d,i}$ matches the color of the searcher not assigned to neither S_p nor S_{-p}, i.e., $x_{d,i} \in X$. Clean each such $L_{d,i}$ by using searchers of color $\mathtt{C}_d$, $\mathtt{R}$ and $x_{d,i}$. Hence, at this point each $L_{d,i}$ for which the literal $l_{d,i}$ is satisfied in C is clean. Place the two searchers of color $\mathtt{C}_d$ in each remaining contaminated $L_{d,i}$ on vertex belonging to A_0. Because at least one subtree $L_{d,i}$ is clean for each $d \in \{1, \ldots, m\}$, two searchers of color $\mathtt{C}_d$ are sufficient. Then, continue cleaning A_0 towards the next v_h.

4. We now describe the moves of the search strategy performed once a vertex v_j, $j \in R$, is reached while cleaning A_0.

   Clean all remaining contaminated edges of S_p and S_{-p} rooted in v_j. Remove the searchers of colors $\mathtt{T}_p, \mathtt{F}_p, \mathtt{V}_p$ when they are no longer necessary.

5. Once v_{a+1} has been reached, clean all remaining contaminated subtrees $L_{d,i}$ (it follows directly from previous steps that searchers of appropriate colors are available) and perform moves as in colorless strategy with the addition of necessary switching of searchers on vertices with multiple colors. Repeat this strategy for each $L_x, x \in \{\mathtt{T}_i, \mathtt{F}_i \mid i \in \{1, \ldots, n\}\}$, and $L'_{x'}, x' \in \{\mathtt{C}_1, \ldots, \mathtt{C}_m\}$.

6. For each color $z \in \mathcal{Q} \setminus \{\mathtt{R}\}$ set a searcher of color z on the common vertex of the subtree H_z and A_0. Then clean all edges incident to each vertex $v_i, i \in L$, which finishes cleaning A_0. The finishing touches of our strategy are simple. Once A_0 is clean, the remaining contaminated parts of subtrees H_z, containing A_i, $i \in L$, can be searched in an arbitrary sequence.

□

Now we will give a series of lemmas that allow us to prove the other implication, namely that a successful monotone strategy implies a valid solution to 3-SAT. The next lemma says that between the first and last moves when all searchers of color R are in an area A_i, no move having all searchers of color R in a different area A_j is possible. The proof is due to a counting argument.

Lemma 4.4.3. No step t_j can occur between any two steps t_i, t'_i:

$$[t_i, t'_i] \cap \left[t_j, t'_j\right] = \emptyset, \quad i \neq j.$$

Proof. The proof is by contradiction. First note that if $t'_i = t_j$ for any $i \neq j$, that would imply that four searchers of color R are present in a graph at once: two in A_i and two in A_j. Hence, we suppose for a contradiction that there exists $j \neq i$ such that $t_i < t_j < t'_i$ or $t_i < t'_j < t'_i$. Consider a step $t \in [t_i, t'_i]$. At least one searcher of color R is in A_i because it is partially clean. If $t \in \{t_j, t'_j\}$, then there are two red searchers in A_j, which contradicts color composition imposed by Lemma 4.4.1. □

We say that a subtree T' is *guarded* by a searcher q on a vertex v if removal of q leads to recontamination of an edge in T'. Note that v does not have to belong to T', or in other words, T' is any subtree of the entire subgraph that becomes recontaminated once the searcher q is removed. We extend our notation to say that T' is *guarded from* T'' if T'' is contaminated and removal of q produces a path that is free of searchers and connects a node of T' with a node of T''.

Informally, the next lemma states the following. Prior and after the moves that have all searchers of color R on A_0, there must be moves having all searchers of color R on some A_i, $i \neq 0$. The argument is due to the fact that we do not have sufficiently many searchers, in total, to guard A_0 from all other A_i's (or, conversely, all other A_i's from A_0).

Lemma 4.4.4. The step t_0 cannot be the first one and t'_0 cannot be the last one in a sequence of steps containing each t_i, $i \in I$, i.e., $\min\{t_i \mid i \in I\} < t_0 \leq t'_0 < \max\{t_i \mid i \in I\}$.

Proof. Suppose for a contradiction that t_0 is the first step, i.e., $t_0 < t_i$, for each $i \in I \setminus \{0\}$. By Lemma 4.4.3, $t_i > t'_0$ for each $i \in I \setminus \{0\}$. Thus, each area A_i contains contaminated edges in step t_0. In step t'_0 all edges of the path P are clean. Hence, P is guarded by at least one searcher from A_i for each $i \in I \setminus \{0\}$. A simple counting argument implies that two searchers of color $\mathtt{V}_p, p \in \{1, \ldots, n\}$, are used — a contradiction. The second case, when t'_0 is the last step, can be argued analogously. □

We draw attention to the two ways of searching A_0 (to which we refer as a folklore). Since A_0 is a caterpillar, one can assume without loss of generality that it is cleaned by $\mathcal{S}$ by the two searchers of color R in the following way. Either, the first searcher of color R is placed, in some move of $\mathcal{S}$, on v_1 and throughout the search strategy it moves along P from v_1 to v_l — we say that such $\mathcal{S}$ *cleans* P *from* v_1 *to* v_l, or the first searcher starts at v_l and moves along P from v_l to v_1 while $\mathcal{S}$

proceeds — we say that such $\mathcal{S}$ *cleans* P *from* v_l *to* v_1. In both cases, the second searcher of color $\mathtt{R}$ is responsible for cleaning edges incident to v_i, $i \in \{1, \ldots, l\}$, when the first searcher is on v_i.

We say that an edge search strategy $\mathcal{S}'$ is a *reversal of* a search strategy $\mathcal{S}$ that consists of l moves if it is constructed as follows: if the move i of $\mathcal{S}$ places (respectively removes) a searcher on a node v, then the move $(l - i + 1)$ of $\mathcal{S}'$ removes (respectively places) the searcher on v, and if the move i of $\mathcal{S}$ slides a searcher from u to v, then the move $(l - i + 1)$ of $\mathcal{S}'$ slides the searcher from v to u. It has been proved in [230] that if $\mathcal{S}$ is a (monotone) edge search strategy, then $\mathcal{S}'$ indeed is a (monotone) edge search strategy. We skip a proof (it is analogous to the one in [230]) that if $\mathcal{S}$ is a monotone search $\tilde{c}$-strategy, then its reversal is also a monotone search $\tilde{c}$-strategy. This allows us to assume the following for the search strategy $\mathcal{S}$ for T_{SAT} we consider in this section:

(*) $\mathcal{S}$ cleans P from v_l to v_1.

Let $R\text{-}pr(v_i), i \in \{1, \ldots, l\}$, be the index of the first move such that two searchers of color $\mathtt{R}$ are in v_i (either at the start or end of the move). Such moves are well defined because the degree of v_i is greater than two. Note that without loss of generality due to (*), $R\text{-}pr(v_l) = t_0$.

Observe that the removal of edges $\{v_a, v_{a+1}\}$ and $\{v_{a+1}, v_{a+2}\}$ from T_{SAT} gives three connected components and let T_{a+1} be the subtree of T_{SAT} that equals the connected component that contains v_{a+1}. For each subtree T', let $Clean(T', t)$ ($Cont(T', t)$, respectively) denote the set of clean (contaminated, respectively) edges in T' immediately prior to the move t.

Intuitively, Lemma 4.4.5 says that when we reach the vertices v_a, v_{a+1}, v_{a+2} while moving along A_0, then the tree T_{a+1} is constructed in such a way that while cleaning it there exists a move in which no searcher is used to guard any A_i with $i \in R$ or any clause component. The configurations of the colors of searchers for the guarding of A_i's and the clause components are those in the family $\mathcal{K}$ below.

Lemma 4.4.5. Let

$$\mathcal{K} = \{\{\mathtt{R}, \mathtt{C}_1\}, \ldots, \{\mathtt{R}, \mathtt{C}_m\}, \{\mathtt{R}, \mathtt{V}_1, \mathtt{T}_1\}, \ldots, \{\mathtt{R}, \mathtt{V}_n, \mathtt{T}_n\}, \{\mathtt{R}, \mathtt{V}_1, \mathtt{F}_1\}, \ldots, \{\mathtt{R}, \mathtt{V}_n, \mathtt{F}_n\}\}$$

For each $K \in \mathcal{K}$, between moves $R\text{-}pr(v_{a+2})$ and $R\text{-}pr(v_a)$, there exists a move t_K which requires all searchers of colors from the set K to be in T_{a+1}.

Proof. An intuition explaining the proof is as follows. Recall that T_{a+1} consists of multiple copies of subtrees L_x and $L'_{x'}$. Arguments are the same for both L_x and $L'_{x'}$. We consider which edges of T_{a+1} are clean in the move $R\text{-}pr(v_{a+2})$: L_x it is either contaminated or contains a guarding searcher. Then we consider which edges of T_{a+1} are clean in the move $R\text{-}pr(v_a)$: in this case L_x it is either clean or contains a guarding searcher. We count how many guarded L_x's can exist in the move $R\text{-}pr(v_a)$. A counting argument reveals that at least three L_x's are clean in the move $R\text{-}pr(v_a)$. From all of the above, these 3 subtrees L_x must have been cleaned after $R\text{-}pr(v_{a+2})$. Among the moves of cleaning 3 subtrees L_x, a move t_K exists.

Consider what can be deduced about $Clean(T_{a+1}, R\text{-}pr(v_{a+2}))$ and $Cont(T_{a+1}, R\text{-}pr(v_{a+2}))$ from (*) and the definition of $R\text{-}pr(v_{a+2})$. Recall that by (*), the strategy we consider cleans P from v_l to v_1. Hence, the edge $\{v_a, v_{a+1}\}$ is contaminated before move $R\text{-}pr(v_{a+2})$. Furthermore, the vertex v_{a+1} cannot be occupied by a searcher during move $R\text{-}pr(v_{a+2})$, because both searchers of color $\mathtt{R}$ are on the vertex v_{a+2}, by the definition of $R\text{-}pr(v_{a+2})$. Therefore, no subtree $L_x, x \in X = \{\mathtt{T}_i, \mathtt{F}_i \mid i \in \{1, \ldots, n\}\}$, or $L'_{x'}, x' \in X' = \{\mathtt{C}_1, \ldots, \mathtt{C}_m\}$, can be fully clean and unguarded in the move $R\text{-}pr(v_{a+2})$, because these subtrees are incident to v_{a+1}. For each subtree L_x, there are two possibilities: either $Clean(L_x, R\text{-}pr(v_{a+2})) = \emptyset$ or $Clean(L_x, R\text{-}pr(v_{a+2})) \neq \emptyset$ in which case L_x contains at least one guarded vertex. The same holds for any $L'_{x'}$.

Next consider what is known about $Clean(T_{a+1}, R\text{-}pr(v_a))$ and $Cont(T_{a+1}, R\text{-}pr(v_a))$. Because v_{a+1} is not guarded in the move $R\text{-}pr(v_a)$ as both searchers of color $\mathtt{R}$ are in vertex v_a, the vertex v_{a+1} is not incident to contaminated edges at this point. Otherwise all edges of P connecting v_{a+1} and v_l would be contaminated which contradicts (*). The spread of contamination at move $R\text{-}pr(v_a)$ from each L_x and $L'_{x'}$ through v_{a+1} can be prevented only in the following way: $Clean(P, R\text{-}pr(v_a))$ is guarded from the contaminated edges in subtrees L_x and $L'_{x'}$ and their copies, and all remaining subtrees in T_{a+1} are clean. Again there are two possibilities: either $Clean(L_x, R\text{-}pr(v_a)) = E(L_x)$ or if $Clean(L_x, R\text{-}pr(v_a)) \neq E(L_x)$, then L_x contains at least one guarded vertex. The same holds for any $L'_{x'}$.

By the above paragraphs, each subtree L_x and $L'_{x'}$ at some point between $R\text{-}pr(v_{a+2})$ and $R\text{-}pr(v_a)$ is either being guarded from or is fully searched. Suppose for a contradiction, that three out of five copies of a subtree L_x or $L'_{x'}$ contain a guarding searcher in the move $R\text{-}pr(v_a)$. A simple counting argument suffices to show that they have to be guarded on a common vertex: for L_x, there are two searchers of color $\mathtt{R}$, and they are placed on the vertex v_a, and one in each of the colors x and $\mathtt{V}_i$ where i is selected so that $x \in \{\mathtt{T}_i, \mathtt{F}_i\}$. Similarly for any $L'_{x'}$ there are only two searchers of color x' which can be placed in it. The only common vertex is v_{a+1} — contradiction with the definition of $R\text{-}pr(v_a)$.

Because we eliminated the option of guarding three subtrees L_x and $L'_{x'}$ at the move $R\text{-}pr(v_a)$, the only remaining possibility is that at least three of the subtrees L_x and $L'_{x'}$ are clean before the move $R\text{-}pr(v_a)$. Consider a move after which the first subtree has been cleaned. This subtree is guarded on v_{a+1} until all edges connected to v_{a+1} are clean, so in subsequent moves at least one of the copies of each L_x and $L'_{x'}$ is cleaned while v_{a+1} is guarded by a searcher of color $\mathtt{R}$. By construction, for each of the following sets: $B \in \mathcal{B} = \{\{\mathtt{R}, \mathtt{V}_1, \mathtt{T}_1\}, \ldots, \{\mathtt{R}, \mathtt{V}_n, \mathtt{T}_n\}\}$ and $F \in \mathcal{F} = \{\{\mathtt{R}, \mathtt{V}_1, \mathtt{F}_1\}, \ldots, \{\mathtt{R}, \mathtt{V}_n, \mathtt{F}_n\}\}$ there exist a subtree L_x requiring searchers in these colors. Note that $\mathrm{es}(L_x) = 3$, so all of those searchers will be required simultaneously in at least one move, whose number is denoted by t_B or t_F respectively, when L_x is being searched. Similarly $L'_{x'}$ will require all searchers of colors $G \in \mathcal{G} = \{\{\mathtt{R}, \mathtt{C}_1\}, \ldots, \{\mathtt{R}, \mathtt{C}_m\}\}$ to be present in T' in a single move, whose number is denoted by t_G. $\mathcal{B} \cup \mathcal{F} \cup \mathcal{G} = \mathcal{K}$, so t_K exists for each $K \in \mathcal{K}$. □

Lemma 4.4.6. Let

$$\mathcal{K}'' = \{\{\mathtt{R}, \mathtt{C}_1\}, \ldots, \{\mathtt{R}, \mathtt{C}_m\}\}$$

For each $K'' \in \mathcal{K}''$, between moves $R\text{-}pr(v_l)$ and $R\text{-}pr(v_{l-2})$, there exists a move t''_K which requires all searchers of colors from the set K'' to be in one of the subtrees $L''_{\mathtt{C}_d}, d \in \{1, \ldots, m\}$. □

We skip the proof because it is analogous to the one of Lemma 4.4.5.

Let $\mathcal{T}$ be the set of subtrees $H_z, z \in \mathcal{Q} \setminus \{\mathtt{R}\}$, and $S_p, S_{-p}, p \in \{1, \ldots, n\}$. For $G \in \mathcal{T}$ let $\tau(G)$ be the index i such that G contains the vertex v_i.

We say in Lemma 4.4.7, informally, that we need to entirely clean all areas A_i with $i \in R$ prior to the part of the search strategy that uses all searchers of color $\mathtt{R}$ on A_0. The latter part is the one that cleans A_0 entirely.

Lemma 4.4.7. The step t_0 is placed in the search sequence in the following way:

$$t_j \leq t'_j < t_0 \leq t'_0$$

for each $j \in R$.

Proof. We first summarize the intuitions used in the proof. We start by using Lemma 4.4.3 and 4.4.4, which give us that A_0 is not the first nor the last area A_i cleaned. We define two sets of numbers (U^- and U^+ below) corresponding to indices of A_i's whose cleaning happens before and after cleaning A_0. During the moves $t_0, \ldots, t'_0$ (i.e., those that clean A_0) the clean subgraph of A_0 has to be guarded from the contaminated A_i's, and the cleaned A_i's have to be guarded from the contaminated part of A_0 Intuitively, once cleaning of A_0 extends past a vertex v_i to which a subtree containing A_i is attached, this A_i needs to be guarded if it's 'status' (being clean or contaminated) is different than that of the area A_0. Consider a move, which we denote by l_{a+1} below in the proof, in which the vertex v_{a+1} divides the clean and contaminated parts of P. We analyze which A_j's, $j \in R$, could have been left contaminated and which ones are guarded in the move l_{a+1}. We obtain that there exists a move $t_K, K \in \mathcal{K}$, described in the Lemma 4.4.5, which can be identified with l_{a+1}. There is not enough searchers to perform t_K while A_j is guarded — a contradiction.

Now we start the formal proof. Define

$$U^- = \{i \in I \mid t_i \leq t'_i < t_0\}$$

and

$$U^+ = \{i \in I \mid t'_0 < t_i \leq t'_i\}.$$

By Lemmas 4.4.3 and 4.4.4, $U^- \neq \emptyset$, $U^+ \neq \emptyset$ and $U^- \cup U^+ = I \setminus \{0\}$. Note that $U^- \cap U^+ = \emptyset$ because the strategy is monotone. Given this notation we restate the lemma as U^- contains all indices of steps $t_j, j \in R$, i.e., $R \subseteq U^-$.

Let $u^- \in U^-$ and $u^+ \in U^+$ be selected arbitrarily. Let $g_{|i|}, i \in I$ be the index of the first move in $[t_0, t'_0]$ such that $v_{|i|}$ is incident to a clean edge. There exists $G \in \mathcal{T}$ such that $\tau(G) = |u^+|$ and G has a contaminated edge between moves

of numbers $g_{|u^+|}$ and t'_0 because $t'_0 < t_{u^+}$. Thus, $Clean(A_0, t), t \in \left[g_{|u^-|}, t'_0\right]$, is guarded from contaminated edges of G.

Let $g'_{|i|}, i \in I$, be the index of the last move in $[t_0, t'_0]$ such that $v_{|i|}$ is incident to a contaminated edge on P. There exists $G' \in \mathcal{T}$ such that $\tau(G') = |u^-|$ and G' has a clean edge between steps t_0 and $g_{|u^-|}$ because $t_{u^-} < t_0$. Thus, $Clean(G', t), t \in \left[t_0, g'_{|u^-|}\right]$, is guarded from contaminated edges of P.

Let l_{a+1} be an arbitrary move number such that the edge $\{v_a, v_{a+1}\}$ is contaminated and the edge $\{v_{a+1}, v_{a+2}\}$ is clean. By $(*)$ such a move exists.

Suppose for a contradiction that $R \nsubseteq U^-$, which is equivalent to $R \cap U^+ \neq \emptyset$. During the move l_{a+1} the following subtrees are guarded:

$$Clean(A_0, l_{a+1}) \quad from \quad A_j, j \in R \cap U^+, \quad because \quad \forall_{j \in R \cap U^+} \quad g_{|j|} < l_{a+1}$$

and

$$Clean(A_i, l_{a+1}) \quad from \quad Cont(A_0, l_{a+1}), i \in L \cap U^- \quad , \\ because \quad \forall_{i \in U^- \cap L} \quad l_{a+1} < g'_{|i|}$$

By Lemma 4.4.5, there exists a move $t_K, K \in \mathcal{K}$, which requires all searchers of colors belonging to K to be present in T_{a+1}. Every edge incident to v_a cannot be clean before the last move which places two searchers of color R in v_a has occurred, therefore $R\text{-}pr(v_a) \leq g'_a$. Two searchers of color R cannot be placed in v_{a+2} before at least one edge incident to it is clean, therefore $g_{a+2} \leq R\text{-}pr(v_{a+2})$. This gives us $g_{a+2} \leq R\text{-}pr(v_{a+2}) < t_K < R\text{-}pr(v_a) \leq g'_a$, and with the fact that in the moves t_K and l_{a+1} the vertex v_{a+1} is occupied, allows us to conclude that for each t_K there exists l_{a+1} such that $t_K = l_{a+1}$.

Let $G \in \mathcal{T}$ be such that $\tau(G) > a+1$. Recall that if $j \in R$ and $\tau(G) = |j|$, then G is isomorphic to some S_p or S_{-p}. Due to construction of S_p and S_{-p}, searcher used to guard $Clean(A_0, l_{a+1})$ from $Cont(G, l_{a+1})$ is in one of the following colors: R, $\mathtt{V}_p$, $\mathtt{T}_p$ if $j > 0$ or R, $\mathtt{V}_p$, $\mathtt{F}_p$ if $j < 0$. Let D_G denote a set of colors in G for each $G \in \mathcal{T}$. Note that for each D_G there exists $K \in \mathcal{K}$ such that $D_G \subseteq K$, and therefore there exists a move t_{D_G} such that all searchers in colors D_G are in T'. Consider a move $t_{D_G} = l_{a+1}$, which requires a searcher of one of the colors in D_G to be present in G, such that it contains an area $A_w, w \in R \cap U^+$. Because T' and $G \in \mathcal{T}$ contain no common vertices, such a move cannot exist — a contradiction with the Lemma 4.4.5. □

The statement of the following lemma is more specific with respect to the previous one: in Lemma 4.4.8 we examine those moves in which each area A_j, $j \in R$, is clean and guarded from A_0. Additionally we are concerned with the colors of searchers guarding these areas. More precisely, between the time when cleaning of A_0 starts and reaches v_b, all subgraphs $A_i, i \in R$, contain clean edges which are guarded from the contaminated edges of A_0. Only searchers of colors R and either $\mathtt{T}_p$ or $\mathtt{F}_p$ can be used for the guarding.

Lemma 4.4.8. In an arbitrary move $t \in [t_0, R\text{-}pr(v_b)]$, each subgraph $Clean(A_j, t)$ for each $j \in R$ is guarded from $Cont(A_0, t)$ by searchers on at least one of these two vertices: a vertex with colors $\{\mathtt{T}_p, \mathtt{R}\}$ in S_p or $\{\mathtt{F}_p, \mathtt{R}\}$ in S_{-p}

Proof. Informally, we analyze the state of the strategy when A_0 is being cleaned but v_b has not been reached, i.e., in a move t from the lemma. In the moves t_0 and $R\text{-}pr(v_b)$ each of S_p and S_{-p} has some clean edges that need to be guarded. In the proof, we consider several cases as to which vertices can be guarded to protect those clean edges in moves t_0 and $R\text{-}pr(v_b)$. Then, we observe that, due to monotonicity, the set of clean edges when the two searchers of color $\mathtt{R}$ are on v_b could not be smaller than in any move prior to it. Thus, if both a vertex with colors $\{\mathtt{T}_p, \mathtt{R}\}$ and a vertex with colors $\{\mathtt{F}_p, \mathtt{R}\}$ need not be guarded between these moves, then a recontamination occurs in the move $R\text{-}pr(v_b)$ which leads to a contradiction.

From Lemma 4.4.7 we have $t_j \leq t'_j < t_0 \leq t'_0$ for each $j \in R$. All stars of color $\mathtt{R}$ in subtrees S_p and S_{-p} attached to vertices $v_{|j|}, j \in R$, contain clean edges before the move t_0, so at the move t_0 each such star is guarded from contaminated edges in A_0.

Consider the moves number t_0 and $R\text{-}pr(v_b)$. In these moves both searchers of color $\mathtt{R}$ are in A_0, and $v_{|j|}$ are not attached to any clean edges, so guarding searchers are still necessary at move $R\text{-}pr(v_b)$. Recall that by the definition of R, $b \geq |j|$. Because searchers of color $\mathtt{R}$ are in A_0, the vertices with the following colors are occupied by searchers: $\mathtt{F}_p$ or $\mathtt{V}_p$ in S_{-p} and $\mathtt{V}_p$ or $\mathtt{T}_p$ in $S_p, p \in \{1, \ldots, n\}$. Because only one searcher of color $\mathtt{V}_p$ is available, at least one other searcher is placed on vertex with either $\mathtt{F}_p$ or $\mathtt{T}_p$ color, denoted by c. In each S_p and S_{-p} there are only two such vertices separated by an edge of color c. It remains to be proven that at least one of them has to be guarded in an arbitrary move $t \in [t_0, R\text{-}pr(v_b)]$.

Assume for a contradiction that both vertices with color $\mathtt{F}_p$ and $\mathtt{T}_p$ are no longer guarded in a move $t \in [t_0, R\text{-}pr(v_b)]$, thus all edges incident to vertices with color c are clean. By monotonicity, all these edges are clean in the move $R\text{-}pr(v_b)$. Edges of color $\mathtt{R}$ incident to the vertex v_i such that $\tau(G) = i$ are contaminated. Since the two searchers of color $\mathtt{R}$ are in v_b in the move $R\text{-}pr(v_b)$, they are not in G. Thus, the searcher of color c is the only one that can be used for guarding $Clean(G, R\text{-}pr(v_b))$ from $Cont(A_0, R\text{-}pr(v_b))$. Since all edges incident to the vertex occupied by this searcher are clean, some recontamination occurs. □

Lemma 4.4.9. Let $x_1, \ldots, x_n$ and a Boolean formula $C = C_1 \wedge C_2 \ldots \wedge C_m$ be an input to 3-SAT. If there exists a search strategy using $2 + 3n + 2m$ searchers for T_{SAT}, then the answer to 3-SAT is $\mathtt{YES}$.

Proof. The proof revolves around the configuration of searchers in the move $R\text{-}pr(v_b)$. We start by recalling the construction of subtrees based on clauses that must have been cleaned up to this point and the colors of searchers required to clean them. Then, we will use Lemma 4.4.8 to address the availability of these colors. We define a Boolean assignment as follows: x_p is true if and only if a searcher of color $\mathtt{T}_p$ does *not* guard the area A_{a+1+p} in the move $R\text{-}pr(v_b)$, otherwise x_p is false.

Let $X \subset \{\mathtt{T}_i, \mathtt{F}_i \mid i \in \{1, \ldots, n\}\}$ denote the colors of those searchers. By Lemma 4.4.8, a valid assignment will occur during execution of arbitrary successful search strategy using $2 + 3n + 2m$ searchers. We argue that some literal in each clause $C_d, d \in \{1, \ldots, m\}$, is true under the above assignment.

By construction of T_{SAT}, $v_h, h \in \{b+1, b+2, \ldots, l-1\}$, is the root of a subtree $L_{d,i}, i \in \{1, 2, 3\}$. $L_{d,i}$ contains an edge of color $\mathtt{T}_p$ if and only if the clause C_d contains a variable x_p, and it contains an edge of color $\mathtt{F}_p$ if and only if C_d contains a variable's negation, $\overline{x_p}$. Let us denote this color as $x_{d,i}$.

Consider the step t_0. It is impossible for any $L_{d,i}$ to be completely clean because all edges incident to $v_h, h \in \{b+1, b+2, \ldots, l-1\}$, are contaminated (there are not enough searchers of color $\mathtt{R}$). Consider the move $R\text{-}pr(v_b) > t_0$. Each $L_{d,i}$ is completely clean or $Clean(A_0, R\text{-}pr(v_b))$ is guarded by searchers of colors $x_{d,i}$ and $\mathtt{C}_d$ because v_h is connected to clean edges and unguarded (there are not enough searchers of color $\mathtt{R}$). There are only two searchers of color $\mathtt{C}_d$, so for each d one of the subtrees $L_{d,i}$ has a searcher of other color or is clean. Because $\mathrm{es}(L_{d,i}) = 3$, three out of five searchers in the following colors can be used: $\mathtt{R}, x_{d,i}, \mathtt{C}_d$. The only searcher which is present in $L_{d,i}$ at move $R\text{-}pr(v_b)$ has color $x_{d,i}$. Without loss of generality we assume that the strategy cleans a subtree if possible before guarding other subtrees rooted in the same vertex. Consider a move $m_{d,i}$ such that $\mathrm{es}(L_{d,i})$ searchers are used in $L_{d,i}$. Due to the way the subtree is colored, a searcher in each color $\mathtt{R}, x_{d,i}, \mathtt{C}_d$ has to be used in order to clean it. After t_0 a searcher of color $\mathtt{R}$ guards clean part of A_0, so only one searcher out of those five, namely the one of color $\mathtt{C}_d$, can be present outside of $L_{d,i}$. $L_{d,i}$ can be fully cleaned only if searcher of color $x_{d,i}$ is available at this point during $[t_0, R\text{-}pr(v_b)]$, or all edges of color $x_{d,i}$ were clean in the move t_0 . Due to its color this searcher can not be used to replace any searcher of color $x_{d,i}$ outside of $L_{d,i}$.

Let us address what follows if an edge of color $x_{d,i}$ was clean in the move t_0. By construction, it can be guarded from contaminated edges of P by a searcher of one of the following colors: $x_{d,i}$, $\mathtt{C}_d$, $\mathtt{R}$ in the moves of numbers from the interval $[R\text{-}pr(v_l), R\text{-}pr(v_{l-2})]$. By Lemma 4.4.6, there exists a move of number in this interval such that all searchers of colors $\mathtt{C}_d$ and $\mathtt{R}$ are not in $L_{d,i}$. Thus, in order to avoid recontamination, clean edges of color $x_{d,i}$ can be guarded only by a searcher of the same color.

Suppose for contradiction that no literal in a clause $C_i, i \in \{1, \ldots, n\}$ is true and a search strategy for T_{SAT} exists. By Lemma 4.4.8, one of the searchers of color $x \in \{\mathtt{T}_p, \mathtt{F}_p\}$, or a searcher of color $\mathtt{R}$ is placed outside of $L_{d,i}$ during $[t_0, R\text{-}pr(v_b)]$. By the definition of a Boolean assignment $x \in X$. Additionally Lemma 4.4.8 guarantees that no searcher of color x is used during cleaning any $L_{d,i}$. If $x = x_{d,i}$ then $m_{d,i}$ can not be performed and $L_{d,i}$ is guarded in the move $R\text{-}pr(v_b)$. Because there are only two searchers of color $\mathtt{C}_d$, in order for a strategy to exist at least one subtree $L_{d,1}$, $L_{d,2}$, $L_{d,3}$ for each d is cleaned before $R\text{-}pr(v_b)$, or all three have to be guarded, and for that to happen they have to contain edges in at least one color $x_{d,i} \neq x$ corresponding to a true $l_{d,i}$ in the clause C_d. □

4.5 NP-hardness of non-monotone searching of trees

This section is devoted to proving the problem remains NP-hard when non-monotone search strategies are allowed:

Theorem 4. The problem HGS is NP-hard in the class of trees.

For the proof, we adapt the tree T_{SAT} described in the previous section. The modified tree is denoted by $\tilde{T}_{\mathrm{SAT}}$ and it is obtained by performing the following operations on the tree T_{SAT}. In order to preserve the familiar notation we denote each component of $\tilde{T}_{\mathrm{SAT}}$ analogous to its counterpart in T_{SAT} with an additional sign $\sim$ above its designation.

We add $4n$ vertices to the path P in the following fashion. Replace the edges $\{v_{a+1}, v_{a+2}\}$ and $\{v_{a+1}, v_a\}$ with paths of color $\mathtt{R}$ of length $2n$ each, denoted by $\tilde{P}_R$ and $\tilde{P}_L$ respectively. Enumerate the vertices of $\tilde{P}$ in $\tilde{T}_{\mathrm{SAT}}$ as $\tilde{v}_i$ in such a way that $\tilde{v}_1 = v_1$, $\tilde{v}_{\tilde{a}+1} = v_{a+1+2n}$, $\tilde{v}_{\tilde{b}} = v_{b+1+4n}$, $\tilde{v}_{\tilde{l}} = v_{l+4n}$. Note that this enumeration preserves the informal division of vertices into sets on the left and right of $\tilde{v}_{a+1}$, and $\tilde{R} = \{\tilde{a}+2, \ldots, \tilde{b}-1\} \cup \{-(\tilde{b}-1), \ldots, -(\tilde{a}+2)\}$ is defined accordingly.

We use $2n$ additional colors $O = \{\mathtt{O}_{1,1}, \ldots, \mathtt{O}_{n,1}, \mathtt{O}_{1,2}, \ldots, \mathtt{O}_{n,2}\}$. For each $o \in O$ create a tree H_o following the construction defined in the previous section and attach one to a unique vertex of the path $\tilde{P}_L$. We do the same for the path $\tilde{P}_R$ so that $4n$ subtrees are created in total. Let $\tilde{H}_o(\tilde{R})$ denote a subtree containing an edge of color $o \in O$ attached to the vertex $\tilde{v}_i, i \in \tilde{R}$.

Next we modify the construction of each subtree $L_x, x \in \{\mathtt{T}_i, \mathtt{F}_i \mid i \in \{1, \ldots, n\}\}$ rooted in v_{a+1} in the following way. Remove 2 leaves of color $\mathtt{R}$ and attach 3 children by the edges of color $\mathtt{O}_{i,1}$ to each leaf. Then attach 3 children by the edges of color $\mathtt{O}_{i,2}$ to each of the new leaves. Finally attach 2 children by the edges of color $\mathtt{R}$ to each of the lastly added leaves. Denote the modified L_x as $\tilde{L}_x$. Note that $\mathrm{es}(\tilde{L}_x) = 5$ and $\tilde{L}_x$ requires searchers of colors $x, \mathtt{R}, \mathtt{V}_i, \mathtt{O}_{i,1}, \mathtt{O}_{i,2}$ to be simultaneously present in some move in $\tilde{L}_x$ in order to search it. In $\tilde{T}_{\mathrm{SAT}}$, eleven copies of $\tilde{L}_x$ are rooted in $\tilde{v}_{a+1}$ in place of five copies of L_x rooted in v_{a+1} in the original T_{SAT}. Whenever an arbitrary copy can be chosen the notation of $\tilde{L}_x$ is used, when the argument requires copies to be distinct they are denoted by $\tilde{L}_{x,i}, i \in \{1, \ldots, 11\}$.

Define a star $O_p, p \in \{1, \ldots, n\}$ with 3 leaves incident to edges of colors: $\mathtt{O}_{p,1}$, $\mathtt{O}_{p,1}$ and $\mathtt{O}_{p,2}$. We modify each subtree $\tilde{S}_p$ in $\tilde{T}_{\mathrm{SAT}}$ corresponding to S_p constructed according to the definition in the previous section. Recall that each S_p and $\tilde{S}_p$ (respectively S_{-p} and $\tilde{S}_{-p}$) contains an edge of color $\mathtt{T}_p$ ($\mathtt{F}_p$ respectively). Define a $plugin(v, u)$ operation for vertices u and v of a tree as replacement of maximal subtree such that u and v are its leaves with a copy of O_p in such a way that u is identified with a leaf of color $\mathtt{O}_{p,1}$ and v is identified with a leaf of color $\mathtt{O}_{p,2}$. For each $\tilde{T} \in \{\tilde{S}_p, \tilde{S}_{-p} \mid p \in \{1, \ldots, n\}\}$ denote the endpoint which belongs to $\tilde{A}_0$ of the edge of color $\mathtt{T}_p$ or $\mathtt{F}_p$ in $\tilde{T}$ as u_1, and the other endpoint of this edge as u_2. Denote the endpoint which belongs to $\tilde{A}_j, j \in \tilde{R}$, of edge of color $\mathtt{V}_p$ as u_3, and the other endpoint of this edge as u_4. Perform $plugin(u_1, u_1)$, $plugin(u_2, u_3)$ and $plugin(u_4, u_4)$.

Informally speaking, the described modification prevents using recontamination to switch the searchers used as a basis for Boolean assignment without undoing all the progress made while cleaning subtrees corresponding to the clauses.

The following lemma follows directly from the lower bound $\beta(\tilde{T}_{\mathrm{SAT}})$ and the proof is analogous to Lemma 4.4.1

Lemma 4.5.1 (Color assignment)**.** A $\tilde{c}$-strategy using $k = 5n + 2m + 2$ searchers has to color them in the following fashion: one searcher for each color in

$$\{\mathtt{T}_p, \mathtt{F}_p, \mathtt{V}_p, \mathtt{O}_{p,1}, \mathtt{O}_{p,2} \mid p \in \{1, \ldots, n\}\}$$

and two searchers for each color in $\{\mathtt{R}\} \cup \{\mathtt{C}_1, \ldots, \mathtt{C}_m\}$. □

Lemma 4.5.2. Let $x_1, \ldots, x_n$ and a Boolean formula $C = C_1 \wedge C_2 \ldots \wedge C_m$ be an input to the 3-SAT. If the answer to 3-SAT is `YES`, then there exists a search strategy using $5n + 2m + 2$ searchers for $\tilde{T}_{\mathrm{SAT}}$.

Proof. We propose a modification to the monotone strategy described in Lemma 4.4.2. Note that the modified strategy is still monotone (we aim to show that recontamination does not help to search $\tilde{T}_{\mathrm{SAT}}$). In the instruction 2 we clean $\tilde{A}_{-(\tilde{a}+2n+1+p)}$ instead of $A_{-(a+1+p)}$ and stars O_p instead of singular edges of color $\mathtt{R}$ replaced by these stars during construction of $\tilde{T}_{\mathrm{SAT}}$. No searcher of color either $\mathtt{O}_{p,1}$ or $\mathtt{O}_{p,2}$ has already been placed on $\tilde{T}_{\mathrm{SAT}}$ so it is always possible. We introduce an additional instruction 2' executed after the instruction number 2.

2'. For each $\tilde{H}_o(\tilde{R})$ place a searcher of color o on the vertex of color o and belongs to an $\tilde{A}_i$ in $\tilde{H}_o(\tilde{R})$. Then, clean each A_i, where $\tilde{v}_i \in \tilde{P}_R$, and then the edge of color o. The searcher of color o stays in $\tilde{H}_o(\tilde{R})$.

In instruction 4 a searcher of color o is removed from $\tilde{H}_o(\tilde{R})$ when the entire $\tilde{H}_o(\tilde{R})$ becomes clean during cleaning of $\tilde{A}_0$ in order to ensure that $\tilde{L}_x$ can be searched. □

4.5.1 Preliminaries on non-monotone strategies for $\tilde{T}_{\mathrm{SAT}}$

Let G' be a subgraph of G. We define a *successful attempt* $S\text{-}Attempt(G', i) = [t, t']$ as a maximal interval of numbers of moves such that for each $j \in [t, t']$, $Cont(G', j) \neq G'$, $Cont(G', j) \neq \emptyset$ and $Clean(G', t') = G'$, and i is the ordeal number of this attempt among other successful attempts on G'. Analogously define an *unsuccessful attempt* $U\text{-}Attempt(G', i) = [t, t']$ as a maximal interval of numbers of moves such that for each $j \in [t, t']$, $Cont(G', j) \neq G'$, $Clean(G', j) \neq G'$ and i is the ordeal number of this attempt among other unsuccessful attempts on G'.

By definition, at least one edge of G' is clean during an attempt on G'. We remove the prefix U or S whenever the success of the attempt is not important at the point of speaking.

Note that any search strategy which cleans a graph G contains the $S\text{-}Attempt(G', 1)$ for any subgraph G', thus in order to show that cleaning a graph is impossible it suffices to prove that there exists a subgraph for which there can be no successful attempt. By strengthening the previous statement we obtain the following:

Observation 4.5.1. If G' is a subgraph of G, then for each *S-Attempt*(G, i) there exists *S-Attempt*(G', j) such that *S-Attempt*$(G', j) \subseteq$ *S-Attempt*(G, i).

We skip the proof of the following lemma as it follows from the folklore of the way to clean a caterpillar graph.

Lemma 4.5.3. If in any move of *Attempt*$(\tilde{P}, i) = [t, t']$ both edges $\{\tilde{v}_1, \tilde{v}_2\}$ and $\{\tilde{v}_{\tilde{l}}, \tilde{v}_{\tilde{l}-1}\}$ are contaminated, then the attempt is unsuccessful. □

As a consequence we can be sure that a *S-Attempt*$(\tilde{P}, i)$ starts cleaning at either $\tilde{v}_1$ or $\tilde{v}_{\tilde{l}}$ and ends at $\tilde{v}_{\tilde{l}}$ or $\tilde{v}_1$ respectively. If it starts at $\tilde{v}_l$ and if the edge $\{\tilde{v}_{j+1}, \tilde{v}_j\}$ is clean, then the next clean edge of $\tilde{P}$ can be only $\{\tilde{v}_j, \tilde{v}_{j-1}\}$ or the next contaminated set of edges of $\tilde{P}$ has to include all edges $\{\tilde{v}_{j+1}, \tilde{v}_j\}, \ldots, \{\tilde{v}_{j+1+x}, \tilde{v}_{j+x}\}$ for some $j + x < l, j - 1 > 0$ and no other edges of $\tilde{P}$. We say that in such attempt a strategy *cleans* $\tilde{P}$ *from* $\tilde{v}_{\tilde{l}}$ *to* $\tilde{v}_1$. Thanks to the result concerning reversal of strategies established in [230] a symmetrical case does not need to be considered. Thus, we establish an assumption about non-monotone strategies analogous to (*):

(**) $\mathcal{S}$ cleans $\tilde{P}$ in *S-Attempt*$(\tilde{P}, 1)$ from $\tilde{v}_{\tilde{l}}$ to $\tilde{v}_1$.

Denote the number of a move when two searchers of color $\mathtt{R}$ arrive on the vertex $\tilde{v}_i$ of the path $\tilde{P}$ in *S-Attempt*$(\tilde{P}, 1)$ as *R-pr*$(\tilde{v}_i, j)$ where j is the ordinal number of the move *R-pr*$(\tilde{v}_i, j)$ among other moves *R-pr*$(\tilde{v}_i, j)$ of index i. Specifically *R-pr*$(\tilde{v}_i, j + 1)$ is the number of the first such move after the move *R-pr*$(\tilde{v}_i, j)$. Informally speaking, the j in the expression *R-pr*$(\tilde{v}_i, j)$ indicates how many times the vertex $\tilde{v}_i$ was reached in the first successful attempt to clean $\tilde{P}$. Let $\tilde{P}_i, i \in \{1, \ldots, \tilde{l}\}$, denote $\tilde{T}_{\mathrm{SAT}}[\{\tilde{v}_{\tilde{l}}, \ldots, \tilde{v}_{\tilde{i}}\}]$. Let $i(j) = k + j$ denote the ordeal number of the *S-Attempt*$(G, i(j))$ such that $i(0)$ is the ordeal number of the first *S-Attempt*(G, k) such that *S-Attempt*$(G, k) \subseteq$ *S-Attempt*$(\tilde{P}, 1)$. Whenever we speak of *S-Attempt*$(G, i(0))$, we are concerned with the first attempt to clean G within the first attempt which successfully cleaned $\tilde{P}$.

4.5.2 Some technical lemmas

In the next lemma we show that before the vertex $\tilde{v}_{\tilde{a}}$ is reached in the first successful attempt to clean $\tilde{P}$, for each of the listed sets of colors there exists a move which requires searchers of those color to be present in $\tilde{T}_{\tilde{a}+1}$. Analogously to Lemma 4.4.5, these sets correspond to sets of colors of subtrees attached to the vertices of $\tilde{P}_{\tilde{a}}$.

Lemma 4.5.4. Let

$$\mathcal{K} = \{\{\mathtt{R}, \mathtt{C}_i\} \mid i \in \{1, \ldots, m\}\} \cup \{\{\mathtt{R}, \mathtt{V}_n, \mathtt{O}_{n,1}, \mathtt{O}_{n,2}, x\} \mid x \in \{\mathtt{T}_1, \ldots, \mathtt{T}_n, \mathtt{F}_1, \ldots, \mathtt{F}_n\}\}.$$

For each $K \in \mathcal{K}$, in *S-Attempt*$(\tilde{P}_{\tilde{a}}, i(0))$, there exists a move $t_K \leq$ *R-pr*$(\tilde{v}_{\tilde{a}}, 1)$ which requires all searchers of colors from the set K to be in $\tilde{T}_{\tilde{a}+1}$.

Proof. The proof is divided into three parts. First we argue that $\tilde{T}_{\tilde{a}+1}$ could not have been left clean before the move *R-pr*$(\tilde{v}_{\tilde{a}+2}, j)$. Then we argue that $\tilde{T}_{\tilde{a}+1}$ has to be clean before move *R-pr*$(\tilde{v}_{\tilde{a}}, 1)$. Finally we analyze the construction of $\tilde{T}_{\tilde{a}+1}$ to show that cleaning the subtrees of $\tilde{T}_{\tilde{a}+1}$ requires certain sets of searchers. These sets of searchers are listed as a family $\mathcal{X}$ and $\mathcal{Y}$.

Let us consider moves performed only in the attempt *S-Attempt*$(\tilde{P}, 1)$ which by (**) cleans $\tilde{P}$ from $\tilde{v}_{\tilde{l}}$ to $\tilde{v}_1$. Choose the minimal j such that *R-pr*$(\tilde{v}_{\tilde{a}+2}, j) \in$ *S-Attempt*$(\tilde{P}_{\tilde{a}}, i(0))$. Hence, the edge $\{\tilde{v}_{\tilde{a}}, \tilde{v}_{\tilde{a}+1}\}$ is not clean at move *R-pr*$(\tilde{v}_{\tilde{a}+2}, j)$ move. The subtree $\tilde{T}_{\tilde{a}+1}$ cannot be completely clean in the move *R-pr*$(\tilde{v}_{\tilde{a}+2}, j)$, because it contains the vertex $\tilde{v}_{\tilde{a}+1}$, which is unoccupied (by the definition of *R-pr*$(\tilde{v}_{\tilde{a}+2}, 1)$) and incident to the contaminated edge $\{\tilde{v}_{\tilde{a}+1}, \tilde{v}_{\tilde{a}}\}$ (by (**)).

On the other hand each copy of $\tilde{L}_x$ has to be either completely clean or guarded in the move *R-pr*$(\tilde{v}_{\tilde{a}}, 1)$. Suppose otherwise for a contradiction, then the contamination spreads unobstructed through $\tilde{v}_{\tilde{a}+1}$, which cannot be occupied by a searcher during move *R-pr*$(\tilde{v}_{\tilde{a}}, 1)$, to $\tilde{v}_{\tilde{l}}$ and, by the Lemma 4.5.3, the attempt *S-Attempt*$(\tilde{P}, 1)$ fails contrary to its definition.

Because the two searchers of color $\mathtt{R}$ are not in a non-leaf vertex of $\tilde{L}_x$ in neither of the moves number *R-pr*$(\tilde{v}_{\tilde{a}}, 1)$ and *R-pr*$(\tilde{v}_{\tilde{a}+2}, j)$ at most four copies of $\tilde{L}_x$ can be guarded at each of these moves. In total at most eight out of eleven copies of $\tilde{L}_x$ can be cleaned only partially between these two moves. Which means that in the attempt *S-Attempt*$(\tilde{P}_{\tilde{a}}, i(0))$ there exists a *S-Attempt*$(\tilde{L}_{x,1} \cup \tilde{L}_{x,2} \cup \tilde{L}_{x,3}, k)$.

By construction, for each of set:

$$X \in \mathcal{X} = \{\{\mathtt{R}, \mathtt{V}_1, \mathtt{O}_{1,1}, \mathtt{O}_{1,2}, x\} \mid x \in \{\mathtt{T}_1, \ldots, \mathtt{T}_n\} \cup \{\mathtt{F}_1, \ldots, \mathtt{F}_n\}\}$$

there exists a subtree $\tilde{L}_x$ requiring searchers of these colors. Note that $\mathrm{es}(\tilde{L}_x) = 5$ and $\mathrm{es}(\tilde{L}_{x,1} \cup \tilde{L}_{x,2} \cup \tilde{L}_{x,3}) = 6$, thus all searchers of colors contained in X will be present on some vertices of $\tilde{L}_{x,1} \cup \tilde{L}_{x,2} \cup \tilde{L}_{x,3}$ simultaneously, in at least one move, whose number is contained in *S-Attempt*$(\tilde{L}_{x,1} \cup \tilde{L}_{x,2} \cup \tilde{L}_{x,3}, k)$. Denote the number of the first such move in this attempt as t_X. Because we consider only moves whose numbers belong to *S-Attempt*$(\tilde{P}, 1)$, one searcher of color $\mathtt{R}$ is present on $\tilde{P}$. In the move t_X this searcher occupies $\tilde{v}_{\tilde{a}}$, therefore $t_K \leq$ *R-pr*$(\tilde{v}_{\tilde{a}}, 1)$.

The same argument can be repeated for any $\tilde{L'}_y$ and the respective set from $Y \in \mathcal{Y} = \{\{\mathtt{R}, \mathtt{C}_i\} \mid i \in \{1, \ldots, m\}\}$ to prove existence of analogously defined t_Y. $\mathcal{K} = \mathcal{X} \cup \mathcal{Y}$ finishes the proof. □

We skip the proof of the following lemma because it is analogous to the one of Lemma 4.5.4.

Lemma 4.5.5. Let

$$\mathcal{K}'' = \{\{\mathtt{R}, \mathtt{C}_i\} \mid i \in \{1, \ldots, m\}\}.$$

For each $K'' \in \mathcal{K}''$, in *S-Attempt*$(\tilde{P}_{\tilde{l}-2}, i(0))$, there exists a move $t_{K''}$ which requires all searchers of colors from the set K'' to be to be in one of the subtrees $\tilde{L''}_{\mathtt{C}_d}, d \in \{1, \ldots, m\}$. □

Let $\tilde{\mathcal{T}}$ be the set of all subtrees $\tilde{S}_p, \tilde{S}_{-p}, \tilde{H}_{\mathtt{0}_{p,1}}(\tilde{R}), \tilde{H}_{\mathtt{0}_{p,2}}(\tilde{R}), p \in \{1, \ldots, n\}$. Recall that these subtrees are attached to the vertices $\tilde{v}_j$ for $j \in \tilde{R}$.

Lemma 4.5.6. In the move *R-pr*$(\tilde{v}_{\tilde{a}}, 1)$ all subtrees in $\tilde{\mathcal{T}}$ are clean.

Proof. Each move t_K introduced in Lemma 4.5.4, where $K \in \mathcal{K}$, happens before the vertex v_a is reached in the first successful attempt to clean $\tilde{P}$. Each subtree in $\tilde{\mathcal{T}}$ contains only vertices of colors found in some $K \in \mathcal{K}$. Every subtree in $\tilde{\mathcal{T}}$ is connected to vertices which were cleaned before v_a. Because in the move t_K all searchers of colors present in some subtree of $\tilde{\mathcal{T}}$ are in T_{a+1} if this subtree of $\tilde{\mathcal{T}}$ contains a contaminated edge, then the attempt to clean $\tilde{P}$ fails — since we analyze a successful attempt, a contradiction occurs. Finally we show that a recontamination of this subtree of $\tilde{\mathcal{T}}$ cannot happen prior to the move *R-pr*$(\tilde{v}_{\tilde{a}}, 1)$.

By Lemma 4.5.4, *R-pr*$(\tilde{v}_a, 1) \geq t_K$ for each $K \in \mathcal{K}$. By construction, for each subtree $\tilde{G} \in \tilde{\mathcal{T}}$ there exists $K \in \mathcal{K}$ such that the set of colors in vertices of $\tilde{G}$, denoted by $Q(\tilde{G})$, is a subset of K. Note that any subtree in $\tilde{\mathcal{T}}$ is connected to the vertex $\tilde{v}_{\tilde{a}+1}$ only by a subpath of $\tilde{P}$, which may contain a subset of the following vertices $\{\tilde{v}_{\tilde{a}+1}, \ldots, \tilde{v}_{\tilde{b}}\}$, and all these vertices are of color R.

Suppose for a contradiction that $\tilde{G}$ contains a contaminated edge in the move t_K such that $Q(\tilde{G}) \subseteq K$. Because all searchers in colors $Q(\tilde{G})$ are in $\tilde{T}_{\tilde{a}+1}$ (one of color R is explicitly on the vertex $\tilde{v}_{\tilde{a}+1}$) and $\tilde{G} \cap \tilde{T}_{\tilde{a}+1} = \emptyset$, $\tilde{G}$ contains no searchers in the move t_K. If it were to contain a contaminated edge at this point, then the contamination would have spread unobstructed along the path $\tilde{P}$ from a vertex by which $\tilde{G}$ is attached (which may be one of the following $\{\tilde{v}_{\tilde{a}+2}, \ldots, \tilde{v}_{\tilde{b}}\}$) to the edge $\{\tilde{v}_{\tilde{l}}, \tilde{v}_{\tilde{l}-1}\}$. By Lemma 4.5.3, the attempt *S-Attempt*$(\tilde{P}, 1)$ fails, which contradicts its definition.

If $\tilde{G}$ contained no contaminated edge nor searchers and was adjacent to the *Clean*$(\tilde{P}, t_K)$), then in the move t_K it was completely clean. By construction, recontamination may be introduced to $\tilde{G}$ only through the vertex of $\tilde{P}$ by which it is attached. Because $t_K \geq$ *R-pr*$(\tilde{v}_{\tilde{a}+2}, j)$ there is always a searcher of color R guarding it from $Cont(\tilde{P}, t), t \in [t_K,$ *R-pr*$(\tilde{v}_{\tilde{a}}, 1)]$, so it stays clean. □

Lemma 4.5.7. There is at least one clean edge in each $\tilde{A}_r, r \in \tilde{R}$, in each move of *S-Attempt*$(\tilde{P}, 1)$.

Proof. Assume for the sake of a contradiction, that an area $\tilde{A}_r$ is fully contaminated in the move *R-pr*$(\tilde{v}_{\tilde{l}}, 1)$. es$(\tilde{A}_r) = 2$ so it cannot be cleaned in *S-Attempt*$(\tilde{P}, 1)$, because at least one searcher of color R is in $\tilde{P}$. This contradicts Lemma 4.5.6, because the move *R-pr*$(\tilde{v}_{\tilde{a}}, 1)$, in which all of the subtrees in $\tilde{\mathcal{T}}$ are clean, belongs to *S-Attempt*$(\tilde{P}, 1)$. □

Let $\tilde{P}^+_{\tilde{b}}$ denote $\tilde{T}_{\mathrm{SAT}}[\{\tilde{v}_{\tilde{l}}, \ldots, \tilde{v}_{\tilde{b}}, v\}]$ where v is a leaf incident $\tilde{v}_{\tilde{b}}$ which does not belong to $\tilde{P}$. Consider the attempt *S-Attempt*$(\tilde{P}^+_{\tilde{b}}, i(0))$. Note that *R-pr*$(\tilde{v}_{\tilde{l}}, 1) \in$ *S-Attempt*$(\tilde{P}^+_{\tilde{b}}, i(0))$, *R-pr*$(\tilde{v}_{\tilde{b}}, 1) \in$ *S-Attempt*$(\tilde{P}^+_{\tilde{b}}, i(0))$. Informally speaking, we define the first successful attempt to clean the subtree containing clause components, which are by construction connected to the vertices of the path $\tilde{P}_{\tilde{b}}$.

Lemma 4.5.8. There is at least one searcher in each $\tilde{G} \in \tilde{\mathcal{T}}$ guarding each $Clean(\tilde{A}_r, t)$ where $r \in \tilde{R}$, $t \in S\text{-}Attempt(\tilde{P}^+_{\tilde{b}}, i(0))$.

Proof. Because the move t belongs to $S\text{-}Attempt(\tilde{P}, 1)$, by Lemma 4.5.7 there is at least one clean edge in $\tilde{A}_r$ which has to be separated from $Cont(\tilde{P}, t)$. By the definition of t, at least one searcher of color $\mathtt{R}$ is on $\tilde{v}_c, c > \tilde{b}$ and edges $\{\{\tilde{v}_{\tilde{b}}, \tilde{v}_{\tilde{b}+1}\}, \ldots, \{\tilde{v}_{\tilde{l}}, \tilde{v}_{\tilde{l}-1}\}\}$ are contaminated. Therefore there is at least one contaminated edge incident to each G, and a searcher guarding $Clean(\tilde{A}_r, R\text{-}pr(\tilde{v}_{\tilde{b}}, k))$ can only be placed in the subtree $\tilde{G} \in \tilde{\mathcal{T}}$ containing $\tilde{A}_r$. □

Note that in $\tilde{T}_{\mathrm{SAT}}$ the areas $\tilde{A}_r, r \in \tilde{R}$, are subgraphs of $\tilde{H}_{o_p}(\tilde{R})$, $\tilde{S}_p$ and $\tilde{S}_{-p}$ (while only S_p and S_{-p} were included in T_{SAT}), hence the two separate lemmas below. The lemmas state that in the moves in which two searchers of color $\mathtt{R}$ are on the path P within the attempt to clean the clause components, only searchers of colors different than $\mathtt{R}$ can be used to guard clean edges of the subtrees $\tilde{H}_{o_p}(\tilde{R})$, $\tilde{S}_p$ and $\tilde{S}_{-p}$.

Lemma 4.5.9. For any i and j, such that $R\text{-}pr(\tilde{v}_i, j) \in S\text{-}Attempt(\tilde{P}^+_{\tilde{b}}, i(0))$, in a move $R\text{-}pr(\tilde{v}_i, j)$ there is a searcher of color $o_p \in \{\mathtt{O}_{p,1}, \mathtt{O}_{p,2}\}, p \in \{1, \ldots, n\}$, in $\tilde{H}_{o_p}(\tilde{R})$.

Proof. By the definition of $R\text{-}pr(\tilde{v}_i, j)$, there can be no searcher of color $\mathtt{R}$ on any vertex of $\tilde{H}_{o_p}(\tilde{R})$. By Lemma 4.5.8, each of them contains a searcher, and by colors of vertices in $\tilde{H}_{o_p}(\tilde{R})$, it is of color o_p. □

Note that Lemmas 4.5.9 and 4.5.10 speak of the same moves, therefore the pool of available searchers is shared between them.

Lemma 4.5.10. For any i and j, such that $R\text{-}pr(\tilde{v}_i, j) \in S\text{-}Attempt(\tilde{P}^+_{\tilde{b}}, i(0))$, in a move $R\text{-}pr(\tilde{v}_i, j)$ there is a searcher of color $\mathtt{T}_p$ (respectively $\mathtt{F}_p$) or $\mathtt{V}_p$ in $\tilde{S}_p$ ($\tilde{S}_{-p}$ respectively).

Proof. By the definition of $R\text{-}pr(\tilde{v}_i, j)$, there can be no searcher of color $\mathtt{R}$ on any vertex of $\tilde{S}_p$. By Lemma 4.5.8, each of them contains a searcher, and by colors of vertices in $\tilde{S}_p$ and Lemma 4.5.9, it is of color $\mathtt{T}_p$ or $\mathtt{V}_p$. Proof for $\tilde{S}_{-p}$ is analogical. □

4.5.3 Adaptation to non-monotonicity — there is no going back

Because of a possibility of recontamination, the previous lemmas are insufficient to obtain a result analogous to that given by Lemma 4.4.8. In this section we find a configuration of searchers that cannot be used in $\tilde{P}^+_{\tilde{b}}$ in a successful attempt to clean $\tilde{P}^+_{\tilde{b}}$, cf. Lemma 4.5.13.

Lemma 4.5.11. At least one of searchers of color from the following set: $Q = \{\mathtt{O}_{p,1}, \mathtt{O}_{p,2}, x_p\}, x_p \in \{\mathtt{T}_p, \mathtt{F}_p\}$, has to remain in each $\tilde{S}_p \cup \tilde{S}_{-p}$ in each move $t \in [R\text{-}pr(\tilde{v}_{\tilde{l}}, 1), R\text{-}pr(\tilde{v}_{\tilde{b}}, 1)] \subseteq S\text{-}Attempt(\tilde{P}^+_{\tilde{b}}, i(0))$.

Proof. The proof revolves around analyzing colors of vertices in $\tilde{S}_p$ and $\tilde{S}_{-p}$ which can be used by guarding searchers in Lemma 4.5.8. A switch is a change in the guarding searchers of colors different than $\mathtt{O}_{p,1}$ or $\mathtt{O}_{p,2}$ in $\tilde{S}_p$. In order to make such a switch, we either have to clean O_p or recontaminate it. We exclude the possibility to clean any star O_p in the moves t listed in the lemma (see Observation 4.5.2). Thus, we have to consider recontamination of O_p. Then, we use Lemma 4.5.10 to establish that such a switch can occur once in $\tilde{S}_p$ (or $\tilde{S}_{-p}$ analogically). Finally we look at guarding searchers in both $\tilde{S}_p$ and $\tilde{S}_{-p}$ in the moves $R\text{-}pr(\tilde{v}_{\tilde{l}}, 1)$ and $R\text{-}pr(\tilde{v}_{\tilde{b}}, 1)$ and show that a switch can occur in either $\tilde{S}_p$ or $\tilde{S}_{-p}$, but not both, between these moves.

Pick a vertex, denoted by $u_{p,t}$, such that $u_{p,t} \in V(\tilde{S}_p)$ ($u_{-p,t} \in V(\tilde{S}_{-p})$ respectively) and it is incident to a contaminated and a clean edge in the move $t \in S\text{-}Attempt(\tilde{P}^+_{\tilde{b}}, i(0))$. Let us focus only on $u_{p,t}$ as the approach is analogous for $u_{-p,t}$. By Lemma 4.5.8, this vertex exists.

Denote the set of colors other than $\mathtt{O}_{p,1}, \mathtt{O}_{p,2}$ in the set of colors of $u_{p,t}$ as $c_{p,t}$, i.e., $c_{p,t} = c(u_{p,t}) \setminus \{\mathtt{O}_{p,1}, \mathtt{O}_{p,2}\}$. Recall that $\tilde{S}_p$ and $\tilde{S}_{-p}$ contain copies of the star O_p. Note that if $c_{p,t} = \emptyset$ then $u_{p,t}$ is a central vertex of such a copy of O_p. Otherwise $|c_{p,t}| = 1$. Let $f^+(t)$ denote the number of the first move such that $f^+(t) \geq t$ and $c_{p,f^+(t)} \neq \emptyset$. Let $f^-(t)$ denote the number of the first move such that $f^-(t) \leq t$ and $c_{p,f^-(t)} \neq \emptyset$.

Observation 4.5.2. By Lemma 4.5.8, no

$$S\text{-}Attempt(O_p, i) \subseteq [R\text{-}pr(\tilde{v}_{\tilde{l}}, 1), R\text{-}pr(\tilde{v}_{\tilde{b}}, 1)]$$

exists.

By construction, any maximal subtree $\tilde{T}$, such that $\tilde{T}$ is a subgraph of $\tilde{S}_p$ where $u_{p,f^-(t)}$ and $u_{p,f^+(t')}, t < t'$, are leaves and $c_{p,f^-(t)} \neq c_{p,f^+(t')}$, contains a copy of O_p, to which we refer further as O'_p. If $c_{p,h} = \emptyset$, then $h \in Attempt(O_p, i)$ and $f^-(h)$ corresponds to the beginning of this attempt ($f^+(h)$ corresponds to its end, respectively). Note that both $c_{p,R\text{-}pr(\tilde{v}_{\tilde{l}},1)} \neq \emptyset$ and $c_{p,R\text{-}pr(\tilde{v}_{\tilde{b}},k)} \neq \emptyset$. By Observation 4.5.2, a pair $u_{p,f^-(t)}$ and $u_{p,f^+(t')}$, such that $t, t' \in [R\text{-}pr(\tilde{v}_{\tilde{l}}, 1), R\text{-}pr(\tilde{v}_{\tilde{b}}, k)]$, which satisfies $c_{p,f^-(t)} \neq c_{p,f^+(t')}$, exists only if $O'_p \subseteq \tilde{T}$ has already been clean in the move $f^-(t)$. Similar argument can be repeated for a pair $u_{p,t}$ and $u_{-p,t'}$ (i.e. vertices in $\tilde{S}_{-p}$ and $\tilde{S}_{-p}$) with the conclusion that a pair which satisfies the above constraints does not exist — there are no clean copies of O_p between them. Informally, we can switch a searcher of color in $c_{p,t} \neq \emptyset$ to a searcher of color in $c_{p,t'} \neq c_{p,t} \neq \emptyset$ only by causing recontamination, and we cannot use the first searcher again. Thus, there is a finite number of switches in $S\text{-}Attempt(\tilde{P}^+_{\tilde{b}}, i(0))$.

Note that by Lemma 4.5.10, $c_{p,R\text{-}pr(\tilde{v}_l,1)}$ ($c_{-p,R\text{-}pr(\tilde{v}_l,1)}$ respectively) contains either $\mathtt{T}_p$ (respectively $\mathtt{F}_p$) or $\mathtt{V}_p$. If $c_{p,t'} = \{\mathtt{R}\}$ we contradict Lemma 4.5.10 in

the next move $R\text{-}pr(\tilde{v}_i, j)$ after t'. Therefore, a set $c_{p,t}$ or $c_{p,t'}$ can contain only a one out of these two colors: $\mathtt{T}_p$, $\mathtt{V}_p$, or be empty. By the previous paragraph and the number of different colors, there exists at most one interval of numbers of moves $J = [f^-(j), f^+(j')]$ such that $c_{p,f^-(j)} = \mathtt{T}_p$ and $c_{p,f^+(j')} = \mathtt{V}_p$ or vice versa. Informally, we can switch the color of required searcher once. The same argument holds for $\tilde{S}_{-p}$ and colors $\mathtt{F}_p$, $\mathtt{V}_p$ Denote the corresponding interval as L.

Because there is only one $\mathtt{V}_p$ searcher at least one of the following is true: $c_{p,R\text{-}pr(\tilde{v}_l,1)} = \{\mathtt{T}_p\}$ or $c_{-p,R\text{-}pr(\tilde{v}_l,1)} = \{\mathtt{F}_p\}$, thus at most one edge of color $\mathtt{V}_p$ in $\tilde{S}_{-p} \cup \tilde{S}_p$ is clean. For the same reason at most one the following is true: $c_{p,R\text{-}pr(\tilde{v}_b,k)} = \{\mathtt{V}_p\}$ or $c_{-p,R\text{-}pr(\tilde{v}_b,k)} = \{\mathtt{V}_p\}$. Thus, in a single strategy at most one of the two intervals J and L exists. If J (L respectively) does not exist, then $c_{p,t} \in \{c_{p,R\text{-}pr(\tilde{v}_l,1)}, \emptyset\}$ ($c_{-p,t} \in \{c_{-p,R\text{-}pr(\tilde{v}_l,1)}, \emptyset\}$ respectively) for each move $t \in S\text{-}Attempt(\tilde{P}^+_{\tilde{b}}, i(0))$. By definition of $u_{p,t}$, only searchers of colors $\mathtt{O}_{p,1}, \mathtt{O}_{p,2}$ and those in $c_{p,t}$ can stay in $\tilde{S}_p$ in each move of $[R\text{-}pr(\tilde{v}_{\tilde{l}}, 1), R\text{-}pr(\tilde{v}_{\tilde{b}}, 1)]$. □

Lemma 4.5.12. If the searcher of color $\mathtt{O}_{p,z}, z \in \{1, 2\}$ is not in in $\tilde{H}_{\mathtt{O}_{p,z}}(\tilde{R})$ in a move $t \in S\text{-}Attempt(\tilde{P}^+_{\tilde{b}}, i(0))$, then a searcher of color $\mathtt{R}$ is in $\tilde{H}_{\mathtt{O}_{p,z}}(\tilde{R})$.

Proof. Because $t \in S\text{-}Attempt(\tilde{P}, 1)$ and by Lemma 4.5.7 at least one searcher has to remain in $\tilde{H}_{\mathtt{O}_{p,z}}(\tilde{R})$. It can be of color $\mathtt{R}$ or $\tilde{H}_{\mathtt{O}_{p,z}}(\tilde{R})$. □

Lemma 4.5.13. There exists a set of searchers of colors $\{\mathtt{R}, \mathtt{O}_{p,1}, \mathtt{O}_{p,2}, x_p \mid p \in \{1, \ldots, n\}\}, x_p \in \{\mathtt{T}_p, \mathtt{F}_p\}$ such that all but one have to remain outside of $\tilde{P}^+_{\tilde{b}}$ in each move $t \in [R\text{-}pr(\tilde{v}_{\tilde{l}}, 1), R\text{-}pr(\tilde{v}_{\tilde{b}}, 1)] \subseteq S\text{-}Attempt(\tilde{P}^+_{\tilde{b}}, i(0))$.

Proof. By Lemma 4.5.11, for each $p \in \{1, \ldots, n\}$ at least one of searchers of color from the following set: $Q = \{\mathtt{O}_{p,1}, \mathtt{O}_{p,2}, x_p\}, x_p \in \{\mathtt{T}_p, \mathtt{F}_p\}$, has to remain in $\tilde{S}_p \cup \tilde{S}_{-p}$ in each move $t \in [R\text{-}pr(\tilde{v}_{\tilde{l}}, 1), R\text{-}pr(\tilde{v}_{\tilde{b}}, 1)]$. Let s denote such a searcher of color other than x_p. If s exists then by Lemma 4.5.12 a searcher of color $\mathtt{R}$ is in $\tilde{H}_{\mathtt{O}_{p,z}}(\tilde{R})$. In a move $t \in [R\text{-}pr(\tilde{v}_{\tilde{l}}, 1), R\text{-}pr(\tilde{v}_{\tilde{b}}, 1)]$ at least one searcher of color $\mathtt{R}$ is on the path $\tilde{P}_{\tilde{b}}$ so s is unique. Then, $(\tilde{S}_p \cup \tilde{S}_{-p}) \cap \tilde{P}^+_{\tilde{b}} = \emptyset$ and $\tilde{H}_{\mathtt{O}_{p,z}}(\tilde{R}) \cap \tilde{P}^+_{\tilde{b}} = \emptyset$ finish the proof. □

To informally summarize, we show that there exists a set of searchers of colors $\{\mathtt{R}, \mathtt{O}_{p,1}, \mathtt{O}_{p,2}, x_p \mid p \in \{1, \ldots, n\}\}, x_p \in \{\mathtt{T}_p, \mathtt{F}_p\}$ of which at most one at a time can take part in cleaning of $\tilde{P}^+_{\tilde{b}}$.

4.5.4 Conclusion

Lemma 4.5.14. Let $x_1, \ldots, x_n$ and Boolean formula $C = C_1 \wedge C_2 \ldots \wedge C_m$ be an input to the 3-SAT problem. If there exists a search strategy using $2 + 5n + 2m$ searchers for $\tilde{T}_{\mathrm{SAT}}$, then the answer to 3-SAT problem is `YES`.

Proof. The proof revolves around the configuration of searchers in the move $R\text{-}pr(\tilde{v}_{\tilde{b}}, i(0))$. We define a Boolean assignment as follows: x_p is true if and only if

a searcher of color $\mathtt{T}_p$ does *not* guard the area $A_{\tilde{a}+1+p}$ in the move $R\text{-}pr(\tilde{v}_{\tilde{b}}, i(0))$, otherwise x_p is false. Let $X \subset \{\mathtt{T}_i, \mathtt{F}_i \mid i \in \{1, \ldots, n\}$ denote the colors of those searchers. By Lemma 4.5.13, a valid assignment will occur during execution of $\tilde{c}$-search strategy using $2 + 5n + 2m$ searchers. We omit the detailed proof in favor of an analogy to the proof of Lemma 4.4.9.

Let $\tilde{T}$ denote the maximal subtree containing $\tilde{v}_{\tilde{l}}$ such that $\tilde{v}_{\tilde{b}}$ is this subtree's leaf. Note that $\tilde{T}$ is isomorphic to its equivalent in T_{SAT}, and monotone strategies are a subset of strategies available in this version of the problem. We focus on proving that the configuration of searchers in the move $R\text{-}pr(\tilde{v}_{\tilde{b}}, 1)$ has properties analogous to those of the configuration in the move $R\text{-}pr(v_b)$ of a strategy for T_{SAT}. Regardless of the moves performed by searchers in a $\tilde{c}$-strategy if a color $x \notin X$, then an edge of color x in $E(\tilde{T})$ which was contaminated before the move $R\text{-}pr(\tilde{v}_{\tilde{l}}, 1)$ remains contaminated in the moves of numbers from the interval $[R\text{-}pr(\tilde{v}_{\tilde{l}}, 1), R\text{-}pr(\tilde{v}_{\tilde{b}}, 1)]$. Thus, configurations which do not correspond to a valid assignment cannot use the searchers of appropriate colors required to guard them and continue cleaning the tree.

All that remains to be addressed is the possibility of these edges being clean before the move $R\text{-}pr(\tilde{v}_{\tilde{l}}, 1)$ (recall that in the proof of Lemma 4.4.9 we used the notion of monotonicity to resolve this issue, here the argument has to be continued). If this was the case they would have to be guarded by at least one searcher in the moves of numbers from the interval $[R\text{-}pr(\tilde{v}_{\tilde{l}}, 1), R\text{-}pr(\tilde{v}_{\tilde{h}}, 1)]$. By Lemma 4.5.5, there exists a move whose number is in this interval such that all searchers of color $\mathtt{R}$ and $\mathtt{C}_d$ are not in $\tilde{L}_{d,1}$, $\tilde{L}_{d,2}$, $\tilde{L}_{d,3}$. Thus, by the colors of vertices of $\tilde{L}_{d,i}$ only the searcher of color x can prevent recontamination of an edge of color x and it cannot be used, by the definition of X. Furthermore, these edges stay contaminated in the move $R\text{-}pr(\tilde{v}_{\tilde{b}}, i(0))$.

We use only the positions of searchers in a move of a specific number, so we are interested in a result, not the process, of a partial cleaning of T_{SAT} and $\tilde{T}_{\mathrm{SAT}}$. Therefore, most arguments from Lemma 4.4.9 can be applied to $\tilde{T}_{\mathrm{SAT}}$. Recall the conclusion of the proof of Lemma 4.4.9. In order for a $\tilde{c}$-strategy for T_{SAT} to exist at least one subtree $L_{d,1}$, $L_{d,2}$, $L_{d,3}$ for each $d \in \{1, \ldots, m\}$ is cleaned before the clean part of A_0 reaches v_b, or all three have to be guarded, and for that to happen they have to contain edges in at least one color corresponding to $l_{d,i}$ in clause C_d. The same is true for a $\tilde{c}$-strategy for $\tilde{T}_{\mathrm{SAT}}$ and its respective counterparts of T_{SAT}. □

4.6 Polynomially tractable instances

If G is a tree then Lemma 4.2.2 gives us a lower bound of $\beta(G)$ on the number of searchers. In this section we will look for an upper bound assuming that there is exactly one connected component per color. With this assumption we show a constructive, polynomial-time algorithm both for HGS and HCGS.

Let $(E_1, \ldots, E_k)$ be the partition of edges of T so that E_i induces the area of color i in T. Observe that this partition induces a tree structure. More formally,

consider a graph in which the set of vertices is $P_E = \{E_1, E_2, \dots E_k\}$ and $\{E_i, E_j\}$ is an edge if and only if an edge in E_i and and edge in E_j share a common junction in T. Then, let $\tilde{T}$ be the BFS spanning tree with the root E_1 in this graph. We write V_i to denote all vertices of the area with edge set E_i, $i \in \{1, \dots, z\}$.

Our strategy for cleaning T is recursive, starting with the root. The following procedure requires that when it is called, the area that corresponds to the parent of E_i in $\tilde{T}$ has been cleaned, and if $i \neq 1$ (i.e., E_i is not the root of $\tilde{T}$), then assuming that E_j is the parent of E_i in $\tilde{T}$, a searcher of color j is present on the junction in $V_i \cap V_j$. With this assumption, the procedure recursively cleans the subtree of $\tilde{T}$ rooted in E_i.

procedure CLEAN(labeled tree T, E_i) ▷ `Clean the subtree of` T `that`
▷ `corresponds to the subtree of` $\tilde{T}$ `rooted in` E_i

1. For each E_j such that E_j is a child of E_i in $\tilde{T}$ place a searcher of color j on the junction $v \in V_j \cap V_i$.
2. Clean the area of color i using $\mathrm{es}(T[V_i])$ searchers. Remove all searchers of color i from vertices in V_i.
3. For each child E_j of E_i in $\tilde{T}$:
   (a) place a searcher of color i on the junction $v \in V_j \cap V_i$,
   (b) remove the searcher of color j from the vertex v,
   (c) call *Clean* recursively with input T and E_j,
   (d) remove the searcher of color i from the vertex v.

end procedure

Lemma 4.6.1. For a given tree $G = (V(G), E(G), c)$, procedure Clean(G, E_1) cleans G using $\beta(G)$ searchers.

Proof. First, observe that the number of searchers used during the execution of procedure Clean is exactly as specified. Indeed, to clean each of the $T[V_i]$ we use $\mathrm{es}(T[V_i])$ searchers of color i and at most one searcher of other colors.

Note that moves (M2) do not cause recontamination. Indeed, the move defined in step 3b of procedure Clean removes a searcher from the node on which another searcher is present, while the move 3d is performed on node v when both subtree and the parent subtree of v are cleaned. This gives the correctness of search strategy produced by procedure Clean. □

We also immediately obtain.

Lemma 4.6.2. If all the strategies used in step 2 of procedure Clean to clean a subtree $T[V_i]$ are monotone, then the resulting $\tilde{c}$-strategy for G is also monotone. □

It is known that there exists an optimal monotone search strategy for any graph [171] and it can be computed in linear time for a tree [186]. An optimal connected search strategy can be also computed in linear time for a tree [10].

Using Lemmas 4.2.2 and 4.6.1 we conclude with the following theorem:

Theorem 5. Let $G = (V(G), E(G), c)$ be a tree such that the subgraph G_j composed by the edges in E_j is connected for each $j \in \{1, 2, \ldots, z\}$. Then, there exists a polynomial-time algorithm for solving problems HGS and HCGS.

4.7 Conclusions and open problems

Recalling our main motivation standing behind introducing this graph searching model, we note that its properties allow for much easier construction of graphs in which recontaminations need to occur in optimal strategies. Our main open question, following the same unresolved one for connected searching, is whether problems HGS and HCGS belong to NP?

Our more practical motivation for studying the problems is derived from modeling physical environments to whose parts different robots have different access. More complex scenarios than the one considered in this work are those in which either an edge can have multiple colors (allowing it to be traversed by all searchers of those colors), and/or a searcher can have multiple colors, which in turns extends its range of accessible parts of the graph. As a way of modeling mobile entities of different types cooperating to solve various computational tasks (of which searching is just one example), heterogeneous mobile agent computing is receiving a growing interest, including fields like distributed computing. Hence, one may ask for different ways of modeling differences between searchers, which may fit potential practical applications.

Chapter 5

Gossiping with energy constraints

Since the problem was informally introduced in Section 3.1.4, let us keep the introduction brief. Just as in Chapter 4, we replace previous, general definitions with formal ones, specific to the central problem of this chapter. Recall that we are interested in an all-to-all communication using mobile agents. In our formulation, such agents act like postmen transporting copies of data packets between the network nodes. Moreover, each agent has some initial energy which it can use for edge traversals and/or exchange with any other agent sharing its position. Finally, the results are obtained for tree networks. Thereby, without further ado, we present gossiping by energy-constrained mobile agents in tree networks, based on the original work of this title [66].

5.1 Outline

Section 5.2 states the problem formally and introduces some notation we use throughout. Then, Section 5.3 gives a technical summary of our algorithmic approach to solving the gossiping problem. It also provides an intuitive sketch of our high-level ideas, with the main one being that gossiping in this particular setting can be partitioned into executing first a carefully constructed convergecast and then executing a broadcast strategy. Section 5.3 finishes with a statement of our main results. Section 5.4 contains a proof of these results. In particular, Section 5.4.1 deals with an analysis of the above-mentioned convergecast. Section 5.4.2 recalls an existing broadcast algorithm that we can use in our gossiping, and gives some properties of broadcast strategies. These properties, both of convergecast and broadcast are then used in Section 5.4.3 to justify that we may start a gossiping strategy by executing the convergecast from Section 5.4.1 followed by a minimum-cost broadcast. We finish, in Section 5.5, with some remarks regarding the complexity for general graphs and open problems.

5.2 Definitions and Preliminaries

5.2.1 Problem Statement

We are given an edge-weighted tree T, with positive weights. Each node of the tree initially contains a unique piece of information that we call a *data packet*. We are also given a set of mobile agents that are placed at some of the nodes of T. Each agent initially possesses some amount of energy. Both the initial locations of the agents and their energy levels are possibly distinct for different agents. We use the term *configuration* to refer to the positions of agents, their energy levels and the locations of data packets. In order to move along the tree, the agents use energy proportionally to the distance traveled, i.e., traversing an edge of weight x decreases the energy level of an agent by x. Moreover, traversing an edge is treated as a single event, that is, an agent cannot stop or turn around in the middle of an edge. Since an agent needs to have a non-negative energy level at any point, this in particular means that edge traversal is possible only if the energy level of an agent is not smaller that the weight of the edge to be traversed. We refer to such agents that use energy for performing movements as *energy-constrained agents*.

Whenever two agents are occupying the same node they may exchange any amount of energy. When an agent visits a node, it deposits there copies of all currently possessed data packets and acquires copies of all data packets currently present at this node. The model is synchronous, i.e., the time is divided into *rounds*. In each round a single *action* takes place — an action is either edge traversal or energy exchange. Consequently, every agent is present at a node at the end and at the beginning of each round. A sequence of actions is called a *strategy*.

The decision problem that we solve in this work is stated as follows:

Problem 1 (Gossiping)**.** Suppose that we are given an n-node tree T, and a set of k energy-constrained agents placed at nodes arbitrarily, each having some (possibly different) initial amount of energy. Does there exist a strategy that results in each node having a copy of data packet of every other node?

In our algorithmic approach we will show that gossiping can be essentially decomposed into solving two easier problems, namely convergecast and broadcast.

Problem 2 (Convergecast)**.** Suppose that we are given an n-node tree T with a selected node r, and an initial configuration of k energy-constrained agents, each having some (possibly different) initial amount of energy. Does there exist a strategy that results in the node r having a copy of the data packet of every other node?

Problem 3 (Broadcast)**.** Suppose that we are given an n-node tree T with a selected node r, and a configuration of k energy-constrained agents, each having some (possibly different) initial amount of energy. Does there exist a strategy that results in each node having a copy of the data packet of r?

A *gossiping strategy* is a strategy that solves the gossiping problem. Likewise, a *convergecast* (respectively *broadcast*) *strategy* is a strategy that solves the convergecast (broadcast, respectively) problem. If, for a given input instance, there exists a strategy solving a particular problem, then for short we say that the problem is *feasible*.

5.2.2 Notation

All strategies that we consider and analyze in this work will be executed on a rooted tree. Thus, we assume from now on that T and its root r are fixed. The choice and meaning of r will be explained later, here we remark that the algorithm is based on an exhaustive search over all possible roots r. Accordingly, we define a relation between nodes $v \prec u$ if v is a descendant of u in T, that is, u lies on the path between v and r. If, additionally, u and v are adjacent, then we write $\mathrm{par}(v) = u$. For any node v of T, we write T_v to denote the subtree of T induced by v and all its descendants. For each edge e of T, $w(e)$ denotes its weight.

Suppose that an agent traverses an edge from u to v. If $u = \mathrm{par}(v)$, then we say that the action is a *downward* traversal. If $\mathrm{par}(v) = u$, then we say that the action is an *upward* traversal. An edge $e = \{u, \mathrm{par}(u)\}$ is said to be *below* an edge e' (or vertex v) if the path from r to $\mathrm{par}(u)$ contains e' (or v). In an analogous fashion, a subtree $T_{\mathrm{par}(u)}$ is said to be *below* an edge e' (or vertex v) if the path from r to $\mathrm{par}(u)$ contains e' (or v).

Definition 5.2.1. If $m_1, \ldots, m_t$ are all rounds in which an agent moves along an edge and e_i is the edge traversed in round m_i, $i \in \{1, \ldots, t\}$, in a strategy $\mathcal{S}$, then the *cost* of the strategy is $\mathrm{cost}(\mathcal{S}) = \sum_{i=1}^{t} w(e_i)$.

We say that a strategy $\mathcal{S}$ solving a particular problem has *minimum cost* if $\mathrm{cost}(\mathcal{S}) \leq \mathrm{cost}(\mathcal{S}')$ for each strategy $\mathcal{S}'$ solving this problem (where the problem may be convergecast, broadcast or gossiping). Equivalently, a minimum cost strategy has the property that at the end of the last round the total amount of energy of all agents is maximized.

Definition 5.2.2. A convergecast strategy which results in all data packets being at a vertex v at the end of the strategy is referred to as *v-convergecast*.

We finish this section with a remark that simplifies further analysis. If, in a configuration of agents, only agents present at r have positive energy levels, then we say for short that *all energy is at r*.

Claim 5.2.1. Any r-convergecast strategy for T can be transformed into one of the same cost and such that upon completion all energy is at the root of T.

Proof. This follows from the fact that agents can communicate globally. In order to solve the convergecast problem, the data packets from all vertices are transferred to r. Thus, there is at least one upward traversal of each edge. The energy not used by an agent that is present at $v \neq r$ when the strategy ends can be transferred

earlier to an agent that ends at a node u such that $v \prec u$. The required action of energy exchange between these agents can be scheduled when a data packet is shared between them and no additional moves need to be introduced. Thus, inductively, the claim follows. □

5.3 Our Approach to Gossiping

For any gossiping strategy $\mathcal{G}$, there exists a first round in which all data packets are present at some node. Let this be the aforementioned node r of the input tree. As mentioned earlier, throughout the paper we work with an assumption that r is known and we deal with this assumption by considering all possible choices. Hence, we provide the following definition which ties this vertex with the structure of a gossiping strategy.

Definition 5.3.1. A *mid-point* of a strategy is the first round at the end of which all data packets are present at r. (By the fact that this is the first such round, r is the first such node.)

For a gossiping strategy, all rounds till the mid-point (inclusively) are referred to as the *convergecast stage* and all remaining rounds are the *broadcast stage*. These terms are well defined. Let $\oplus$ be an operator of *concatenation* of strategies. We will be interested in representing a gossiping strategy $\mathcal{G}$ as a concatenation of its convergecast stage $\mathcal{C}$ and its broadcast stage $\mathcal{B}$, i.e., $\mathcal{G} = \mathcal{C} \oplus \mathcal{B}$. In such case the initial configuration in $\mathcal{C}$ is the same as the one in $\mathcal{G}$ and the initial configuration in $\mathcal{B}$ is the final configuration in $\mathcal{C}$. Note that by definition, the mid-point is the last round of the convergecast stage. We now argue that the mid-point indeed splits a gossiping strategy into the two stages, in other words, the actions following the mid-point form a broadcast strategy.

Claim 5.3.1. Any broadcast stage is a broadcast strategy from the root r.

Proof. Recall, that we consider only tree networks, hence, there exists only one path between any two nodes. Assume for the sake of contradiction that the actions following the mid-point do not form a broadcast strategy. Thus, there exists a child u of r such that some data packet that is originally at $v \notin V(T_u)$ arrives at u prior to the mid-point, and does not arrive at u after the mid-point. Since there exists only one path between any two nodes and initially each node contains a unique data packet, all packets initially not present in T_u have traversed the edge $\{u, r\}$, in particular the above-mentioned packet from v. Because the mid-point has not happened yet when the data packet from v arrives at u through the edge $\{u, r\}$, r is missing at least one packet, in particular from T_u. The latter is due to the fact that if r is missing a packet from another subtree $T_{u'}$, $u' \neq u$, then this packet needs to be delivered to u through the edge $\{u, r\}$ after the mid-point and that would also deliver the data packet from v to u after the mid-point, which we have already excluded. In order to deliver the data packet from T_u to r, the edge $\{u, r\}$ has to be traversed. Thus, u contains all data packets before r does — a contradiction with the definition of the mid-point. □

Definition 5.3.2. If, in a gossiping strategy $\mathcal{G}$, at the mid-point all energy is at the root and the r-convergecast stage of $\mathcal{G}$ has minimum cost over all r-convergecast strategies, then we say that $\mathcal{G}$ is *structured*. The above-mentioned r-convergecast stage of such a structured gossiping strategy is in the following denoted by $\mathcal{C}_{\min}$.

Our first main result of this work is the following structural property of the gossiping problem. Informally, it says that in order to find a gossiping strategy, provided that one exists, one should start by computing the r-convergecast strategy $\mathcal{C}_{\min}$.

Theorem 6. If there exists a gossiping strategy $\mathcal{G}$ for a given tree T, then there exists a structured gossiping for T. Moreover, the strategy $\mathcal{C}_{\min}$ can be computed in linear time.

Theorem 6 follows directly from Lemma 5.4.3 and Corollary 5.4.1 obtained in Section 5.4.1 and Lemma 5.4.10 in Section 5.4.4.

It will be convenient to be able to compare two strategies, in particular two r-convergecast strategies. We will treat two such strategies as *identical* if two conditions hold:

- for each edge e, both strategies perform the same number of upward edge traversals and the same number of downward edge traversals,
- the final configurations are identical in both strategies (neglecting a possible permutation of agents).

Note that identical r-convergecast strategies have the same cost and the number of upward and downward edge traversals determines the position of the agents (up to a possible permutation). Two such strategies may differ with the order of performing some actions and may differ in the amounts of energy exchanged in particular rounds. For two identical strategies $\mathcal{S}$ and $\mathcal{S}'$ we write $\mathcal{S} \simeq \mathcal{S}'$. For two strategies $\mathcal{S}$ and $\mathcal{S}'$ which make the same edge traversals we write $\mathcal{S} \approx \mathcal{S}'$ and call them *similar*. Note that this is a weaker notion with respect to the operator '$\simeq$', since in the latter we do not insist on having the same final configurations, i.e., different energy levels of agents may occur in addition to their different permutations. Furthermore if any two strategies make the same traversals, that implies that the final positions of agents are the same in both strategies.

Using this notation we adopt the following approach to the gossiping problem. First, we determine the node r, the root. After selecting the root, we compute the r-convergecast strategy $\mathcal{C}_{\min}$ that makes up the convergecast stage of our desired gossiping strategy. Finally, we compute a broadcast from the root. See Figure 5.1 for an example.

To implement the above algorithm, we need to work out certain details. First, we do not know what is the right choice of the root. Note that some choices of r may possibly make the convergecast stage to have smaller cost but may potentially make the broadcast stage that follows more expensive. Since we do not know how to balance that, we will just consecutively try all possible candidates for the root.

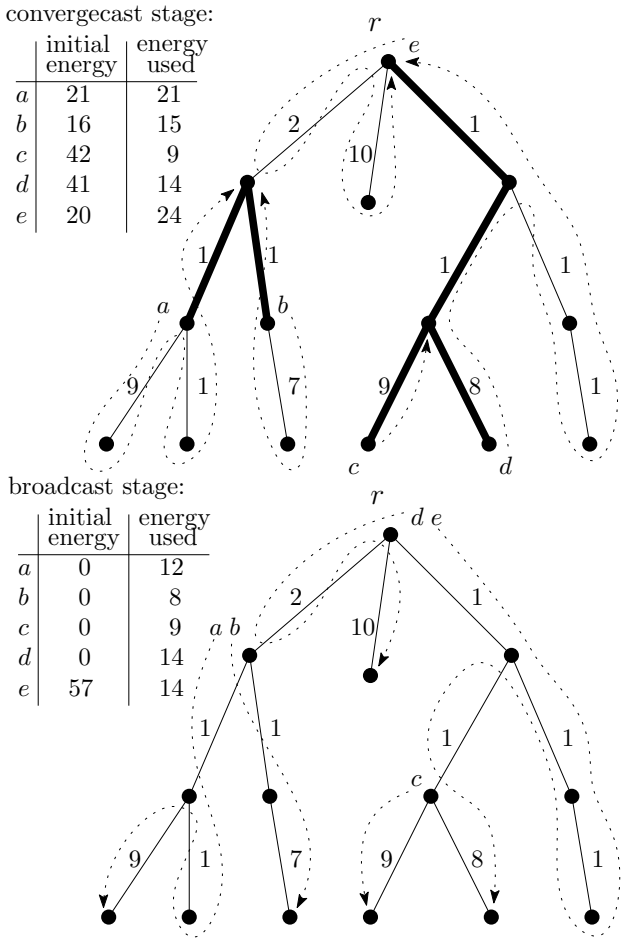


Figure 5.1: A gossiping strategy that starts with an r-convergecast (on the left) and finishes with a broadcast stage (on the right) for a given choice of the root. The illustration of the convergecast stage gives the initial configuration of agents (their initial energy levels are given in the table), their movements (as dotted lines). Some edges are traversed once (we will call them *green* in Section 5.4.1 and they are shown as heavy edges), and some edges are traversed twice (they will be *red* and are depicted as regular ones). Denoting $(x_1, \ldots, x_5) = (a, b, c, d, e)$, the agent x_i is performing all its movements prior to the agent x_{i+1}, $i \in \{1, \ldots, 4\}$ in the convergecast stage. The broadcast stage is illustrated on the right by showing the configuration of agents and their energy levels (in the table) at the mid-point, which is the final configuration of the r-convergecast, and by giving the walks of the agents. In the broadcast stage, the agent x_i performs its walk prior to the agent x_{i-1} for each $i \in \{2, \ldots, 5\}$.

Second, we need to prove that $\mathcal{C}_{\min}$ is indeed a good candidate to be a prefix of a successful gossiping, provided that the correct root has been chosen. At some point, see Corollary 5.4.1, we will prove that $\mathcal{C}_{\min}$ is *unique*, with respect to the operator '$\simeq$' and all energy being at the root at the end of the strategy, and that we can compute it in linear time, see Lemma 5.4.3. We finally obtain (see Theorem 6) that if gossiping is feasible, then there exists a structured gossiping strategy solving it. Thus, the above tells us how to efficiently compute a convergecast stage for which there exists some broadcast stage. The latter one is obtained, as a black-box, on the basis of [68].

This leads to the following theorem, our second main result of the work, whose proof that follows the above sketch is provided in Section 5.4.

Theorem 7. Suppose that an edge-weighted n-node tree T and k arbitrarily placed energy-constrained agents are the input to the gossiping problem. There exists a $O(k^2n^2)$-time algorithm that finds a minimum-cost gossiping strategy whenever gossiping is feasible for this input.

5.4 The Gossiping Algorithm

Convergecast as a separate problem has been solved in [67] but for a different model. The difference in the models is that in our case the agents can finish movements and exchange energy only at nodes, while in [67] they can stop in the middle of edges and thus also exchange energy there. Note that our model is stronger in the sense that the agents are more restricted in their possible actions. For this reason, we may not use the convergecast algorithm given in [67] as a black-box. Moreover, as a second reason, we cannot use an arbitrary r-convergecast to be a prefix of a structured gossiping but instead we need a specific strategy $\mathcal{C}_{\min}$ that, as we conclude in Corollary 5.4.1, is unique. Taking into account the above, we develop a required convergecast algorithm in Section 5.4.1.

5.4.1 The Convergecast Stage

Our convergecast algorithm, called CONVERGECAST and described below, takes an edge-weighted rooted tree with a configuration of energy-constrained agents as an input. The algorithm iteratively removes nodes and edges from the tree using two rules. The first rule triggers a depth-first search (DFS) traversal of a subtree T_v rooted at v (note that this is not the entire subtree rooted at v in the initial input tree but possibly a smaller subtree due to the fact that some edges have been possibly previously removed); the edges removed by this rule are *red*. The second rule corresponds to the situation when an edge that is removed is traversed only upwards in the corresponding convergecast strategy and so this edge will be colored *green* for further reference. See Figure 5.1 for an illustration of red and green edges. If, upon completion of the algorithm, some edges are not removed, then this implies that convergecast is not feasible for the given input. (Note that it may be feasible for another choice of the root.)

In the statement of the pseudo-code we use the following notation. We write energy(a) to denote the energy of an agent a at a given point of strategy construction. In other words, energy(a) refers to the current energy level. We extend this notation by writing energy(T') for any subtree T' of T and this refers to the sum of the current energy levels of all agents located at the nodes of T'.

Algorithm CONVERGECAST: An algorithm for computing a minimum-cost r-convergecast strategy for a given tree T rooted at r.

```
1  for each v ∈ V(T) in postorder do
2      a ← any agent at v ▷ if there is no agent at v then continue
         to the next iteration
3      let a collect all energy from agents present at v
4      if energy(T_v) is at least twice the weight of all edges in T_v and
         E(T_v) ≠ ∅ then
5          let a perform a DFS traversal of T_v ▷ these edges will be red
6          for each agent a' met by a during the DFS traversal do
7              all energy of a' is passed to a
8          remove all edges and nodes below v from the tree
9      if energy(a) is at least the weight of {v, par(v)} and E(T_v) = ∅ then
10         move a from v to par(v) ▷ this edge will be green
11         remove v and the edge {v, par(v)} from the tree
12 return the corresponding r-convergecast strategy or failure if the tree is
     non-empty
```

For an analysis of Algorithm CONVERGECAST we introduce some additional notation. Any r-convergecast strategy $\mathcal{C}$ *induces* the following coloring of the edges of the input rooted tree T:

- if during the execution of $\mathcal{C}$ an edge e is traversed only once, then the color of e is *green* (note that in such case this is an upward traversal),
- otherwise the color of the edge e is *red*.

Note that this coloring is consistent with the colors assigned by Algorithm CONVERGECAST.

Whenever for any tree T and an initial configuration of agents, convergecast is not feasible, then we define a *deficit of* T to be minimum amount of energy x such that, adding at the root of T an agent with x units of energy results in an instance for which convergecast is feasible. For convenience we refer to the two segments of code within the 'if' instructions of Algorithm CONVERGECAST as follows: the code executed in the 'if' statement in lines 4-8 is the *red case* and the code in the 'if' statement in lines 9-11 is the *green case*. Both cases may occur in a single iteration due to the following observation: although at the beginning of an iteration we exclusively have either $E(T_v) = \emptyset$ or $E(T_v) \neq \emptyset$, the red case triggered

when $E(T_v) \neq \emptyset$ may remove all edges and nodes below v resulting in $E(T_v) = \emptyset$ and thus triggering the green case in the same iteration. Also note that it might happen that only one case occurs or none of them occurs in a single iteration of the main loop of the algorithm. We say that the node v of T that we refer to in an iteration of Algorithm CONVERGECAST is the *processed node* in the iteration.

Observation 5.4.1. If, in a coloring induced by a strategy returned by Algorithm CONVERGECAST, an edge is red, then this edge is traversed exactly twice in the strategy: one upward and one downward traversal. □

We now state a technical lemma that will be used twice in the following proofs.

Lemma 5.4.1. Consider an iteration of Algorithm CONVERGECAST in which a red case occurs for a processed node v. For any $u \in V(T_v) \setminus \{v\}$,

$$\text{energy}(T_u) < w(\{u, \text{par}(u)\}) + 2 \cdot \sum_{e \in E(T_u)} w(e). \tag{5.1}$$

Proof. If there is no agent in the subtree T_u, then Inequality (5.1) immediately holds. We prove (5.1) by contradiction assuming that there is an agent at some node of T_u. If there is no agent on any node in $V(T_u) \setminus \{u\}$, then there is an agent on u and by assumption this agent has at least energy(T_u) units of energy. But then according to lines 9-11 of Algorithm CONVERGECAST, the edge $\{u, \text{par}(u)\}$ has been removed in an earlier iteration — a contradiction. Thus, there exists $u' \in V(T_u) \setminus \{u\}$ occupied by an agent. Since (5.1) is violated for u, it is violated for at least one subtree T_x, where x is a child of u. Thus, by repeating this argument, we arrive at a node u' with an agent a' such that (5.1) is violated for u'. Since u' has been processed prior to v, in this earlier iteration, the agent a' performed a DFS traversal of $T_{u'}$. Second, once the DFS performed by a' is completed, the energy level of a' that is present at the node u' is at least $w(\{u', \text{par}(u')\})$, which means that the green case occurs in the iteration that processes u'. This however contradicts the fact that the edge $\{u', \text{par}(u')\}$ still belongs to T_v. Thus, (5.1) holds. □

In the next lemma we prove that the actions performed in the red case are correct, i.e., the DFS can be indeed performed by an agent a. Thus, the lemma implies that each node ordering providing a DFS traversal is feasible, that is, the agent performing the traversal has sufficient energy level to make each particular move.

Lemma 5.4.2. If, in an iteration of Algorithm CONVERGECAST, energy(T_v) is at least twice the weight of all edges in T_v, then any DFS traversal performed by an agent a selected in the iteration is feasible.

Proof. Consider an arbitrary iteration of the main 'for' loop in Algorithm CONVERGECAST in which a node v is processed. We may assume without loss of generality that in all preceding iterations the red case has been executed successfully, i.e., the corresponding DFS traversal is feasible. In particular, what needs

to be shown is that any DFS traversal is feasible because the algorithm does not impose any restrictions on the order of selecting edges by the DFS.

Suppose that the agent a enters a subtree T_u of T_v, $u \prec v$, by traversing the edge $\{u, v\}$ (downward traversal). We will argue that any DFS traversal of T_u together with the traversals along the edge $\{v, u\}$ results in the agent having strictly less energy (this value is denoted by y below) than prior to the traversal. Let x be the energy level of a just before the traversal of this edge. Suppose that a completes any DFS traversal of T_u and returns to $v = \text{par}(u)$ having y units of energy. Lemma 5.4.1 implies that

$$y = x - \left(2w(\{u, v\}) + 2 \cdot \sum_{e \in E(T_u)} w(e) \right) + \text{energy}(T_u) < x.$$

Note that the above choice of u is arbitrary. Thus, in a red case triggered when processing v, if a arrives at any node u during the DFS traversal, then the traversal of the subtree T_u indeed decreases the initial energy level of a, since we also take into account the energy necessary to go back to $\text{par}(u)$. This decrease of energy is the same regardless of the DFS order of visiting the nodes of T_u. Hence, whether the DFS is feasible or not depends only on the overall energy level of a and all agents in the subtree.

To finish the argument, note that the red case is triggered when $\text{energy}(T_v)$ is at least twice the weight of all edges in T_v (see line 4 of Algorithm CONVERGECAST). As we proved, going from v to any child u, doing the DFS traversal of T_u and then returning back to v, which is a part of the DFS selected in line 5, only decreases the energy level of the agent performing the DFS. Since this holds for an arbitrary u, the condition on the energy level from line 4 ensures that the agent has the missing amount of energy when it arrives at v (prior to the downward traversal of $\{v, u\}$). By the missing energy we refer to the energy needed to traverse the edge $\{v, u\}$ twice (first downward and then upwards) and to perform the DFS traversal of T_u. □

Lemma 5.4.3. Algorithm CONVERGECAST correctly determines in linear time whether r-convergecast is feasible. Moreover, if it returns an r-convergecast strategy then its cost is minimum over all r-convergecast strategies for T rooted at r.

Proof. We prove, by induction on the tree size, the following invariant regarding the strategy $\mathcal{C}$ computed by Algorithm CONVERGECAST. For each subtree T_v: if, while processing v, a green or red case occurs (or both), then it holds:

(a) If r-convergecast is feasible for the subtree T_v (with the initial configuration for T restricted to T_v), then $\mathcal{C}$ restricted to T_v is a minimum cost v-convergecast for T_v.

(b) If v-convergecast is not feasible for the subtree T_v (with the initial configuration for T restricted to T_v), then $\mathcal{C}$ has an action in which an agent a moves

from $\mathrm{par}(v)$ to v having, at the end of this action, at least $x+w(v,\mathrm{par}(v))$ units of energy, where x is the deficit of T_v. Note that the arrival of a with the provided energy makes the instance restricted to T_v feasible for v-convergecast.

We prove the claim by following the cases in the pseudo-code of CONVERGECAST. For the base case, if v is a leaf in T, then the claim trivially follows. Thus, suppose that v is not a leaf. By assumption, we consider only nodes at which red or green case occurs.

Consider any child u of v. The removal of the edge $\{u,v\}$ may occur either in a red case or in a green case.

Suppose first that it occurs in a green case. Note that we are analyzing the green case in the iteration when u is processed. Note that a green case only occurs if $E(T_u)=\emptyset$, i.e., all edges below u have been previously removed. The removal of all edges of T_u prior to this iteration implies that u-convergecast in the subtree restricted to T_u is feasible. Thus, by the inductive assumption ((a)), the algorithm performs a minimum-cost convergecast in T_u. Note that if there are more agents at u with non-zero energy at the beginning of the iteration for u that we consider, then due to line 3 of the algorithm, the agent a receives all their energy. Then, the green case dictates the following: the agent a moves from u to v and all remaining agents in T_u have no energy left. This, and the fact that the edge $\{u,v\}$ is traversed only once in $\mathcal{C}$, imply that $\mathcal{C}$ restricted to a tree obtained by attaching the edge $\{u,v\}$ to T_u has minimum cost.

Now suppose that the edge $\{u,v\}$ has been removed in a red case. Since a red case occurs while processing v, that means that an agent is present at v at the beginning of the iteration when v is processed. While analyzing this red case, we write T_v to refer to the subtree at the beginning of the iteration. By the condition of the red case, $E(T_v)\neq\emptyset$. Note that if an agent a' is present at any node u' in $V(T_u)\setminus\{u\}$ then its energy level is not sufficient to traverse the subtree at u', if $E(T_{u'})\neq\emptyset$. Otherwise, the red case condition would be satisfied when u' was processed. This in particular implies, by the inductive assumption ((b)), that each edge of T_v requires a downward traversal in any r-convergecast strategy. By Lemma 5.4.2, any DFS does the job. This proves the minimality of $\mathcal{C}$ restricted to the tree induced by v and nodes u such that $u\prec v$ in the original tree T. □

In the rest of the work we use the symbol $\mathcal{C}_{\mathrm{alg}}$ to denote the r-convergecast strategy computed by Algorithm CONVERGECAST.

Lemma 5.4.4. If, in the coloring induced by an r-convergecast strategy $\mathcal{C}_{\mathrm{alg}}$ computed by Algorithm CONVERGECAST, an edge is red, then the edge is red in any r-convergecast strategy for T.

Proof. Assume for a contradiction that there exists an edge $e=\{u,v\}$, $v=\mathrm{par}(u)$, that is red in the coloring induced by $\mathcal{C}_{\mathrm{alg}}$ but is green in the coloring induced by a different r-convergecast strategy $\mathcal{C}'$. Furthermore, we can assume that e is the lowest edge with such properties.

Let s be the total energy of all agents initially placed in T_u. For the edge e to be green in the coloring induced by $\mathcal{C}'$, the cost c of $\mathcal{C}'$ restricted to the subtree T_u

is at most $s - w(e)$. By Lemma 5.4.3, for edge $\{u', \text{par}(u')\} \in E(T_u)$ that is green in the coloring induced by $\mathcal{C}'$, we have that the cost of $\mathcal{C}'$ in $T_{u'}$ is not smaller than the cost of $\mathcal{C}_{\text{alg}}$ in $T_{u'}$. Obtain a subtree T' by taking T_u and removing the edge $\{u', \text{par}(u')\}$ and all nodes and edges of $T_{u'}$ for all edges $\{u', \text{par}(u')\}$ that are green in the coloring induced by $\mathcal{C}'$. Thus, all edges of T' are red in the coloring induced by $\mathcal{C}'$. The strategy $\mathcal{C}'$ hence needs to traverse each edge of T' twice. Note that we can apply Lemma 5.4.1 for $\mathcal{C}'$ because since it traverses each edge of T' twice, it is the same as $\mathcal{C}_{\text{alg}}$ for the subtree T'. Hence by Lemma 5.4.1, the total energy in T' is insufficient for a double traversal of each edge of T' and an upwards traversal of e — a contradiction. □

Observation 5.4.2. At the end of Algorithm CONVERGECAST all energy is at the root.

Proof. By line 3 of Algorithm CONVERGECAST, an agent performing the traversals of edges collects all energy from agents on the vertex from which it departs and, by line 7, from those agents it meets during its DFS traversal. Clearly all vertices are visited in a convergecast strategy and all energy can be accessed in such manner. Since the energy is transferred by the same agents which carries the data packets, all energy ultimately ends up at r. □

By Observations 5.4.1 and 5.4.2 and Lemmas 5.4.3 and 5.4.4 we obtain.

Corollary 5.4.1. If r-convergecast is feasible then the r-convergecast of minimum cost such that all energy is at the root at the end is unique and $\mathcal{C}_{\min} \simeq \mathcal{C}_{\text{alg}}$. □

5.4.2 The Broadcast Stage

In order to make our main claim regarding the structure of gossiping strategies, namely that we indeed can start a feasible gossiping by executing $\mathcal{C}_{\min}$, we recall in this section a claim regarding the broadcast stages. The claim is a lemma from [68] that gives a structural characterization of the broadcast stages we need to consider: with respect to each edge, any broadcast strategy is partitioned into three steps (formally listed below). This gives a foundation for comparing a gossiping strategy that our algorithm computes with an arbitrary gossiping strategy of minimum cost in Section 5.4.3.

We extend our earlier notation of r-convergecast to gossiping strategies. An *r-gossiping* strategy is one with the property that all data packets are present at the root at the end of the mid-point. Note that the convergecast stage of an r-gossiping strategy is by definition an r-convergecast. We denote by $\mathcal{G}_{\text{alg}}$ a gossiping strategy that we are able to compute in the following way. The $\mathcal{G}_{\text{alg}}$ is an r-gossiping strategy of minimum cost such that $\mathcal{G}_{\text{alg}} = \mathcal{C}_{\min} \oplus \mathcal{B}_{\text{alg}}$ for some broadcast stage $\mathcal{B}_{\text{alg}}$. (Recall that by Corollary 5.4.1, $\mathcal{C}_{\min} \simeq \mathcal{C}_{\text{alg}}$.) Note that this concatenation and cost-minimality of $\mathcal{G}_{\text{alg}}$ imply that the cost of $\mathcal{B}_{\text{alg}}$ is minimal among all possible broadcast stages that follow the r-convergecast $\mathcal{C}_{\min}$.

To compute the broadcast stage $\mathcal{B}_{\text{alg}}$ we use, as a black-box, the following result. Although the algorithm we refer to is slightly more general, i.e., it finds all

nodes from which a broadcast is feasible, we will use it to determine whether the broadcast is feasible from one particular node r.

Theorem 8 ([68])**.** Given an n-node tree rooted at r and an arbitrary initial configuration of k agents, there exists a $O(k^2 n)$-time algorithm that computes a broadcast strategy of minimum cost or determines that no such strategy exists.

Now we analyze the structural properties of the broadcast stages. It is shown in [68] that with respect to any edge $e = \{v, \mathrm{par}(v)\}$, a broadcast strategy of minimal cost might consist of three *steps* (that occur in this order), defined as sequences of the following actions.

1. If the total energy available inside T_v is sufficiently large, an agent will first traverse the edge e upwards bringing some energy to the node $\mathrm{par}(v)$. Such energy may be subsequently transferred to the root r. Depending on the distribution of energy inside T_v this step may or may not exist.

2. An agent a will traverse the edge e downwards in order to transport the data packets into T_v.

3. Then, the third step is one of the following.

   ($\downarrow$) Either a number of other agents traverse the edge e downward in consecutive rounds. Then all these agents, together with the agent a and the agents initially present inside T_v will transport the data packets to all nodes of T_v.

   ($\uparrow$) Or, a number of other agents traverse the edge e upward in consecutive rounds. Before exiting from T_v these agents together with the agent a and the agents initially present inside T_v will transport the data packets to all nodes of T_v.

   ($-$) Or, no other agent traverses e. In this case the agent a together with the agents initially present inside T_v will transport the data packets to all nodes of T_v, eventually terminating their walks inside T_v.

From now on we assume that each optimal broadcast we consider follows the above scheme of Steps 1-3. Note that in case of broadcast stages that do not have Step 1 the only upward traversal of an edge is present in the description of Step 3($\uparrow$).

Observation 5.4.3. The order of actions in any optimal broadcast strategy can be modified in such a way that all moves of Step 1 can be made before any move of Step 2 (and consequently Step 3).

Proof. Assume for a contradiction that Step 2 on some e' is required to make Step 1 on an edge e. This requirement can be only due to lack of resources, either agents or energy, in the subtree T_v, where $e = \{v, \mathrm{par}(v)\}$. To deliver these resources a downward traversal of e is required before Step 1 on e. Contradiction with the definition of Step 1. □

5.4.3 Concatenation of Convergecast and Broadcast

In this section we prove that, provided that r-gossiping is feasible in a given tree, there is an r-gossiping strategy that starts with $\mathcal{C}_{\min}$, i.e., has $\mathcal{C}_{\min}$ as its r-convergecast stage. To that end we compare two r-gossiping strategies, one of which is $\mathcal{G}_{\text{alg}} = \mathcal{C}_{\min} \oplus \mathcal{B}_{\text{alg}}$. The other strategy for our comparison, denoted in the remaining part of the paper by $\mathcal{G}_{\text{opt}}$, is an r-gossiping strategy of minimum cost. Let $\mathcal{C}_{\text{opt}}$ be its convergecast stage. Thus, the choice of $\mathcal{G}_{\text{opt}}$ does not put any configuration restrictions on the midpoint, ensuring that we have a strategy that minimizes the cost over all r-gossiping strategies.

We extend our previous notation by denoting by end-energy($\mathcal{S}$) the sum of energy levels of all agents at the end of a strategy $\mathcal{S}$. The next lemma essentially says that for r-gossiping strategies that use the same amount of energy for their r-convergecast stages and satisfy some technical properties, it is always advisable to start with an r-convergecast after which all energy is at the root. Formally, we have the following.

Lemma 5.4.5. Consider two r-gossiping strategies $\mathcal{G} = \mathcal{C} \oplus \mathcal{B}$ and $\mathcal{G}' = \mathcal{C}' \oplus \mathcal{B}'$ such that:

(a) cost($\mathcal{B}$) and cost($\mathcal{B}'$) are minimized, and

(b) neither in $\mathcal{B}$ nor in $\mathcal{B}'$ Step 1 occurs, and

(c) the positions of agents at the mid-points of $\mathcal{G}$ and $\mathcal{G}'$ are the same, and

(d) in the final configuration of $\mathcal{C}$ all energy is at the root.

If end-energy($\mathcal{C}'$) = end-energy($\mathcal{C}$), then cost($\mathcal{G}$) $\leq$ cost($\mathcal{G}'$).

Proof. Take an arbitrary edge $e = \{v, \text{par}(v)\}$. By assumption (b), no agent at v changes its position in the strategy $\mathcal{B}$ until an agent a arrives at v through the edge e in Step 2. So, a can distribute the amounts of its energy among the agents at v. Thus, the broadcast stage $\mathcal{B}$ following $\mathcal{C}$ can make the same edge traversals as the broadcast stage $\mathcal{B}'$ following $\mathcal{C}'$. The latter is ensured by our assumptions (b) (applied to $\mathcal{B}'$) and (c) that the agents have the same positions at the end of $\mathcal{C}$ and $\mathcal{C}'$. This implies that cost($\mathcal{B}$) $\leq$ cost($\mathcal{B}'$). Since end-energy($\mathcal{C}'$) = end-energy($\mathcal{C}$), it holds cost($\mathcal{C}'$) = cost($\mathcal{C}$). Thus, we obtain

$$\text{cost}(\mathcal{G}) = \text{cost}(\mathcal{C}) + \text{cost}(\mathcal{B}) \leq \text{cost}(\mathcal{C}') + \text{cost}(\mathcal{B}') = \text{cost}(\mathcal{G}'). \qquad \square$$

In the remainder of this section we prove that cost($\mathcal{G}_{\text{alg}}$) $\leq$ cost($\mathcal{G}_{\text{opt}}$). We prove this fact by using Lemma 5.4.5 and thus in this proof we argue that all assumptions of Lemma 5.4.5 hold. Denote $\mathcal{G}_{\text{opt}} = \mathcal{C}_{\text{opt}} \oplus \mathcal{B}_{\text{opt}}$.

We use Lemma 5.4.4 to argue that, informally speaking, if an agent is at different nodes in the final configurations of $\mathcal{C}_{\text{opt}}$ and $\mathcal{C}_{\min}$, then the agents performed in $\mathcal{C}_{\text{opt}}$ some additional edge traversals with respect to $\mathcal{C}_{\min}$. Consider the coloring induced by the strategy $\mathcal{C}_{\min}$. (Recall that $\mathcal{C}_{\min} \simeq \mathcal{C}_{\text{alg}}$ by Corollary 5.4.1.) If an

edge e is green, then by definition $\mathcal{C}_{\min}$ performs only an upward traversal of e and such an upward traversal must occur in each feasible r-convergecast, in particular in $\mathcal{C}_{\text{opt}}$. If an edge e is red, then by the statement of Algorithm CONVERGECAST, e is traversed once in each direction in $\mathcal{C}_{\min}$. By the same argument as for a green edge, e must be clearly traversed upwards in $\mathcal{C}_{\text{opt}}$. It also must be traversed downwards because by Lemma 5.4.4 there is no convergecast strategy that makes e green. In other words, within the subtree T_v, where $e = \{v, \text{par}(v)\}$, the initial configuration does not allow for delivering all data packets from T_v to $\text{par}(v)$ using only the energy of agents initially present in T_v. Thus, a downward traversal of e needs to occur in $\mathcal{C}_{\text{opt}}$.

Let M denote the set of the edge traversals that do occur in $\mathcal{C}_{\text{opt}}$ but do not occur in $\mathcal{C}_{\min}$. We now argue that these additional moves can be ruled out from the convergecast stage of $\mathcal{G}_{\text{opt}}$. In particular, we modify $\mathcal{C}_{\text{opt}}$ into an r-convergecast denoted by $\mathcal{C}'_{\text{opt}}$ and consequently $\mathcal{B}_{\text{opt}}$ into a broadcast $\mathcal{B}'_{\text{opt}}$ in such a way that the cost of a new gossiping strategy $\mathcal{G}'_{\text{opt}} = \mathcal{C}'_{\text{opt}} \oplus \mathcal{B}'_{\text{opt}}$ satisfies $\text{cost}(\mathcal{G}'_{\text{opt}}) = \text{cost}(\mathcal{G}_{\text{opt}})$. Because M are made in addition to the traversals of $\mathcal{C}_{\min}$ we can reschedule them in such a way that each of them happens after all data packets have been gathered at r, i.e., at the end of the mid-point of $\mathcal{G}'_{\text{opt}}$. Thus, M are now a part of $\mathcal{B}'_{\text{opt}}$.

Using our notation, we obtain $\mathcal{C}'_{\text{opt}} \approx \mathcal{C}_{\min}$ (i.e. are similar) and $\mathcal{B}'_{\text{opt}}$ executes first the actions in M (whose order is kept) and then the strategy $\mathcal{B}_{\text{opt}}$. This in particular ensures that Condition (c) of Lemma 5.4.5 holds for $\mathcal{B} := \mathcal{B}_{\text{alg}}$ and $\mathcal{B}' = \mathcal{B}'_{\text{opt}}$.

Observation 5.4.4. For each edge e, the move in Step 2 on e occurs after the move in Step 2 on each edge e' such that e is below e'. □

Observation 5.4.5. The order of actions in $\mathcal{B}'_{\text{opt}}$ can be modified in such a way that for each edge e, all moves of Step 1 below an edge e can be made before any move on and above e. Furthermore, the order of the moves of Step 1 on the same depth of the tree is arbitrary.

Proof. Recall that, by definition, no energy or agents move downwards in any move of Step 1. By Observation 5.4.3, no agents move downwards before all moves of Step 1 are complete. So, the resources necessary to make a move on any edge e are potentially provided only by moves of Step 1 from a subtree below e. □

Because Observation 5.4.5 relays on Observation 5.4.3, and Observation 5.4.5 modifies the order of moves in such a way that it keeps the property that no move in Step 1 happens after Step 2, both of these observations can be applied simultaneously.

Intuitively, the following observation says that, in the presence of Step 3(↑) and Step 3(−), it is possible to conduct the same actions on and below an edge e provided that Step 2 along e gives in $\mathcal{S}'$ not less energy than in $\mathcal{S}$ for the subtree below e.

Observation 5.4.6. Consider two broadcast strategies $\mathcal{S}$ and $\mathcal{S}'$ such that:

- the initial configuration of agents below an edge $e = \{u, \mathrm{par}(u)\}$ is the same in $\mathcal{S}$ and $\mathcal{S}'$, and
- in $\mathcal{S}$ the following moves are done along e: Steps 1, 2, 3($\uparrow$) or 3($-$), and
- Step 2 on e in $\mathcal{S}'$ delivers no less energy to T_u than Step 2 in $\mathcal{S}$.

Then, the same actions following Step 2 below and on e can be made in both strategies regardless whether Step 1 on e exists in $\mathcal{S}'$.

Proof. This is due to the fact that Step 1 only takes away resources from a subtree. □

The next claim is a counterpart of Observation 5.4.6 that considers the remaining case of Step 3($\downarrow$) occurring in $\mathcal{S}'$.

Observation 5.4.7. Consider two broadcast strategies $\mathcal{S}$ and $\mathcal{S}'$ such that:

- the initial configuration of agents below an edge $e = \{u, \mathrm{par}(u)\}$ is the same in $\mathcal{S}$ and in $\mathcal{S}'$, and
- in $\mathcal{S}$ the following moves are done on e: Steps 1, 2, 3($\downarrow$), and
- the agents deliver no less energy to T_u through e in $\mathcal{S}'$, and
- there are no less agents at u after Step 3 in $\mathcal{S}'$ than in $\mathcal{S}$.

Then, the same actions following Step 2 below and on e can be made in both strategies regardless whether Step 1 on e exists in $\mathcal{S}'$.

Proof. This is due to the fact that Step 1 only takes away resources from a subtree. □

5.4.4 Retracing Step 1

We now argue that Step 1 does not need to occur in an optimal broadcast strategy, i.e., we show that given $\mathcal{B}'_{\mathrm{opt}}$ we can construct a broadcast strategy without Step 1. To this end we describe an algorithm that uses $\mathcal{B}'_{\mathrm{opt}}$ as input and use Observations 5.4.6 and 5.4.7 to verify that the obtained strategy is correct.

The algorithm relies on the observation that, since Step 1 is a single upward traversal, it can have two functions in a broadcast: (i) bringing energy to nodes higher up, (ii) bringing an agent to another subtree. It is shown that (i) can be delegated to another agent in the preceding convergecast stage and (ii) can be incorporated into Step 3 by moving the agent's upwards edge traversal to Step 3. If an agent would have to replace the agent which departed from the subtree in Step 1, e.g. only (i) was the original purpose, then this upward traversal is canceled with another agent's downwards edge traversal in Step 3.

We do this inductively for all edges $e_1, \ldots, e_{n-1}$ in a specific order. In order to obtain this ordering we use Breadth First Search on the tree T starting from the

root and for each vertex we take edges connecting its children with the following priority, depending on actions on these edges in $\mathcal{B}'_{\text{opt}}$ (we say that an edge is one of the following *Cases*):

- Case 1: Steps 1, 2, 3(↑)
- Case 2: Steps 1, 2, 3(−)
- Case 3: Steps 2, 3(↑)
- Case 4: Steps 1, 2, 3(↓)
- Case 5: Steps 2, 3(↓)
- Case 6: Step 2, 3(−)

The goal of the given order is to ensure that in a constructed strategy the resources are delivered to vertices before they are used. In particular it is important that the Cases which do not decrease the number of agents in a subtree (i.e. Cases 1, 2 and 3) precede the Cases which do (i.e. Cases 4, 5 and 6). This property is used to support the argument leading to Equations (5.9) and (5.10). Furthermore we note that the order of Cases 1 and 2 could be swapped and the order of Cases 4, 5 and 6 could be arbitrary.

As a preliminary, we modify $\mathcal{B}'_{\text{opt}}$ and the convergecast stage $\mathcal{C}'_{\text{opt}}$ preceding it in order to obtain $\mathcal{C}^0_{\text{opt}}$ and $\mathcal{B}^0_{\text{opt}}$, which will serve as basis for our inductive changes eventually leading to elimination of Step 1 from $\mathcal{G}'_{\text{opt}}$. Specifically, the final configuration of $\mathcal{C}^0_{\text{opt}}$ defines the initial configuration of $\mathcal{B}^0_{\text{opt}}$, which will also be altered in the process. Without considering these changes to $\mathcal{C}'_{\text{opt}}$ the removal of Step 1 in $\mathcal{B}'_{\text{opt}}$ may cause energy deficits in the latter. We note that in the process of starting with $\mathcal{G}^0_{\text{opt}}$ and obtaining the intermediate strategies, their corresponding convergecast strategies are not necessarily the strategy $\mathcal{C}_{\min}$ but the convergecast stage of the final gossiping will be the required $\mathcal{C}_{\min}$.

Lemma 5.4.6. Consider the r-gossiping strategy $\mathcal{G}'_{\text{opt}} = \mathcal{C}'_{\text{opt}} \oplus \mathcal{B}'_{\text{opt}}$. There exists an r-gossiping $\mathcal{G}^0_{\text{opt}} = \mathcal{C}^0_{\text{opt}} \oplus \mathcal{B}^0_{\text{opt}}$ such that:

1. $\mathcal{G}^0_{\text{opt}} \approx \mathcal{G}'_{\text{opt}}$,
2. $\mathcal{C}^0_{\text{opt}} \approx \mathcal{C}'_{\text{opt}}$,
3. in $\mathcal{B}^0_{\text{opt}}$, the moves of Step 1 make the same traversals as in $\mathcal{B}'_{\text{opt}}$ and are taken on edges in the following order: $e_{n-1}, \ldots, e_1$,
4. in the mid point of $\mathcal{G}^0_{\text{opt}}$ at most one agent on each vertex $v \neq r$ has energy. This energy is equal to the amount needed to make the moves of Step 1 by this agent in $\mathcal{B}'_{\text{opt}}$ (and consequently in $\mathcal{B}^0_{\text{opt}}$).

Proof. By Claim 5.2.1 and the fact that $\mathcal{C}'_{\text{opt}}$ is a convergecast strategy, $\mathcal{C}'_{\text{opt}}$ can be modified in such a way that an arbitrary amount of unused energy is delivered to the root with no additional edge traversals. Therefore, a strategy $\mathcal{C}^0_{\text{opt}}$ can be

constructed, such that $\mathcal{C}'_{\text{opt}}$ and $\mathcal{C}^0_{\text{opt}}$ are similar, but less energy than in $\mathcal{C}'_{\text{opt}}$ is left to agents remaining at vertices other than r. We are concerned specifically with the agents which make moves of Step 1 in $\mathcal{B}'_{\text{opt}}$.

The strategy $\mathcal{C}^0_{\text{opt}}$ is obtained as follows. Consider an arbitrary edge $e = \{v, \text{par}(v)\}$ such that Step 1 exists on e in $\mathcal{G}'_{\text{opt}}$. Let a_v be the agent which traverses e in Step 1. Recall that for each edge there exists only one such agent. By the order of steps, a_v does not meet any agent coming from r prior to arrival at v in the strategy $\mathcal{C}'_{\text{opt}}$. The feasibility of these traversals depends solely on energy left in T_v in the convergecast stage. Hence, we build $\mathcal{C}^0_{\text{opt}}$ by taking the same moves on e as $\mathcal{C}'_{\text{opt}}$ but reducing the amount of energy possessed by a_v. At the end of $\mathcal{C}^0_{\text{opt}}$ the agent a_v has only the amount of energy required to traverse e, so that it can make Step 1 on e in $\mathcal{B}^0_{\text{opt}}$, and the energy it transfers to $a_{\text{par}(v)}$ for the Step 1 traversals of the edges above e. Note that $a_{\text{par}(v)}$ and a_v might be the same agent. The remaining energy is collected by an agent which makes an upward move on e and ultimately transferred to r. We stress the fact that the described moves of Step 1 are not made in $\mathcal{C}^0_{\text{opt}}$, in fact no additional moves are. This ensures that points 2 and 4 of the lemma are true and can be used in an argument about any strategy which makes the same traversals in Step 1 as $\mathcal{B}'_{\text{opt}}$.

Consider a sequence of actions following $\mathcal{C}^0_{\text{opt}}$ and denoted by $\mathcal{B}$. Next, we show that $\mathcal{B}'_{\text{opt}}$ and $\mathcal{B}$ are similar despite their different initial energy distribution. Since each agent a_v has enough energy to traverse the corresponding edge in Step 1, $\mathcal{B}$ can make the same traversals in Step 1 as $\mathcal{B}'_{\text{opt}}$. Furthermore, their order in $\mathcal{B}$ is altered with respect to $\mathcal{B}'_{\text{opt}}$ in such a way that they are made before any move of Step 2.

The argument for Steps 2 and 3 is more complex. We say that agents *meet* when they are at the same vertex at the same time. Let R_0 be the set of agents at r in the final configuration of $\mathcal{C}^0_{\text{opt}}$. $R_0 \neq \emptyset$ because $\mathcal{C}^0_{\text{opt}}$ is a convergecast strategy. R_t is defined recursively as follows: in any given round t let $R_t = R_{t-1} \cup P$, where P is the set of agents which meet an agent $a \in R_{t-1}$ in round t. Informally, R_t is the set of all agents which met up to round t an agent that was at r in the final configuration of $\mathcal{C}^0_{\text{opt}}$. We argue that we can order the traversals in $\mathcal{B}$, in such a way that in any given round t only agents in R_t or agents making moves of Step 1 traverse edges. Hence, energy distributed as described in point 4 can be exchanged among agents in R_t in order to make each edge traversal of $\mathcal{B}$ possible, making it a broadcast strategy.

Consider agents on the vertex u of an edge $e = \{u, \text{par}(u)\}$. By Observation 5.4.4, the move of Step 2 in $\mathcal{B}'_{\text{opt}}$ on an arbitrary edge $e' = \{v, \text{par}(v)\}$ occurs after the move of Step 2 on every edge of the path from r to v. Consequently, if e' is below e, then only moves of Step 1 can be made in $T_{\text{par}(v)}$ up to this point. Thus, no agent on u can move downwards before Step 2 on e. Furthermore, only one agent on v can move upwards in Step 1 before the move of Step 2 on e. Therefore, all but one agent remain stationary and meet the agent making the move of Step 2 on e. Hence, every agent with all data packets which moves at round t has met either an agent in R_t, or is in R_t itself, or an agent which made some moves of Step 1, collected all data packets from a vertex, and then made at least one

move of Step 2 without meeting an agent in R_t. Since in $\mathcal{B}$ all moves of Step 1 have been done before any move of Step 2, the agents which made the moves of Step 1 are stationary until they meet an agent from R_t. Thus, the last possibility is eliminated from $\mathcal{B}$.

Finally, in order to obtain $\mathcal{B}^0_{\text{opt}}$ we modify $\mathcal{B}$ in such a way that moves of Step 1 in $\mathcal{B}^0_{\text{opt}}$ are taken on edges in the following order: $e_{n-1}, \ldots, e_1$. This in particular ensures that point 3 is true. By Observation 5.4.5, such a modification can be made by altering only the order of moves. Furthermore, the property that in $\mathcal{B}$ all moves of Step 1 has been done before any move of Step 2 is preserved in $\mathcal{B}^0_{\text{opt}}$.

To summarize, for both pairs of strategies: $\mathcal{C}^0_{\text{opt}} \approx \mathcal{C}'_{\text{opt}}$ and $\mathcal{B}^0_{\text{opt}} \approx \mathcal{B}'_{\text{opt}}$, which ensures that $\mathcal{G}^0_{\text{opt}} \approx \mathcal{G}'_{\text{opt}}$. □

We now define a strategy $\mathcal{B}^i_{\text{opt}}$ for each $i \in \{1, \ldots, n-1\}$. Consider the set of edges $\{e_1, ..., e_i\}$, $i \leq n-1$. Let a vertex $u \in e_k = \{u, \text{par}(u)\}, k \leq i$, such that there is no $e_j, j \leq i$, below u be called a *terminal* vertex of $\mathcal{B}^i_{\text{opt}}$. Let the subtree T_u attached to a terminal vertex u be called an *auxiliary subtree* of $\mathcal{B}^i_{\text{opt}}$. We hope that this abuse of notation will make the arguments easier to follow.

Recall that $\mathcal{B}^0_{\text{opt}}$ and its initial configuration, i.e., the one at the end of $\mathcal{C}^0_{\text{opt}}$, were outlined in Lemma 5.4.6. Let $\mathcal{B}^i_{\text{opt}}$ for $i > 0$ be a sequence of actions in a subtree, denoted as $\widetilde{T}_i$, consisting of $e_1, ..., e_i$ (called the *core subtree* of $\mathcal{B}^i_{\text{opt}}$) and all auxiliary subtrees of $\mathcal{B}^i_{\text{opt}}$. Let the edge-induced subtree by $E(T) \setminus E(\widetilde{T}_i)$ be called the *leftover subtree* of $\mathcal{B}^i_{\text{opt}}$. We will omit the index whenever the subtree being referred is unambiguous. In the core subtree of $\mathcal{B}^i_{\text{opt}}$ the actions performed by $\mathcal{B}^i_{\text{opt}}$ will be a subset of actions in the core subtree of $\mathcal{B}^{i-1}_{\text{opt}}$ and modified actions from $\mathcal{B}^0_{\text{opt}}$ on e_i. In the auxiliary subtrees there will be either no actions, or the actions will be the same as in $\mathcal{B}^0_{\text{opt}}$ (up to their placement in the whole strategy). An edge $e_a = \{u, \text{par}(u)\}$ is said to be the *younger* (*older*) than $e_b = \{v, \text{par}(v)\}$ if $a > b$ (respectively $a < b$) and $\text{par}(u) = \text{par}(v)$. If no such edge e_b exists, then e_a is the *youngest* (*oldest*). See Figure 5.2 for an overview of these terms.

Before we dive into the description of $\mathcal{B}^i_{\text{opt}}$'s, we define the strategy $\mathcal{C}^i_{\text{opt}}$ for each $i \in \{1, \ldots, n-1\}$ announced earlier that gives the initial configuration for $\mathcal{B}^i_{\text{opt}}$. Let $\mathcal{C}^i_{\text{opt}}$, such that $\mathcal{C}^i_{\text{opt}} \approx \mathcal{C}^0_{\text{opt}}$, be a convergecast strategy such that in its final configuration at most one agent on each vertex $v \neq r$ has energy. This energy is equal to the amount needed to make the moves of Step 1 by this agent in $\mathcal{B}^i_{\text{opt}}$. Note that $\mathcal{C}^i_{\text{opt}}$ does not make these moves (recall that these are the moves in M defined at the beginning of this section that constitute the moves of Step 1 that we intend to eliminate). We also remark that once we claim that the final strategy $\mathcal{B}^{n-1}_{\text{opt}}$ has no moves of Step 1, we obtain that all energy is at the root in $\mathcal{C}^{n-1}_{\text{opt}}$ and hence $\mathcal{C}^{n-1}_{\text{opt}}$ equals $\mathcal{C}_{\text{alg}}$ (recall that $\mathcal{C}^0_{\text{opt}} \approx \mathcal{C}'_{\text{opt}}$, by Lemma 5.4.6). This fact is important because we need to keep the property that $\mathcal{C}_{\text{alg}}$ can always lead to an optimal gossiping.

Let $\downarrow_e(\mathcal{S})$ be the number of downward traversals of an edge e in a strategy $\mathcal{S}$. Let $\uparrow_e(\mathcal{S})$ be the number of upward traversals of an edge e in a strategy $\mathcal{S}$. If $\mathcal{S}$ is defined for a subtree of T and e does not belong to the subtree, then

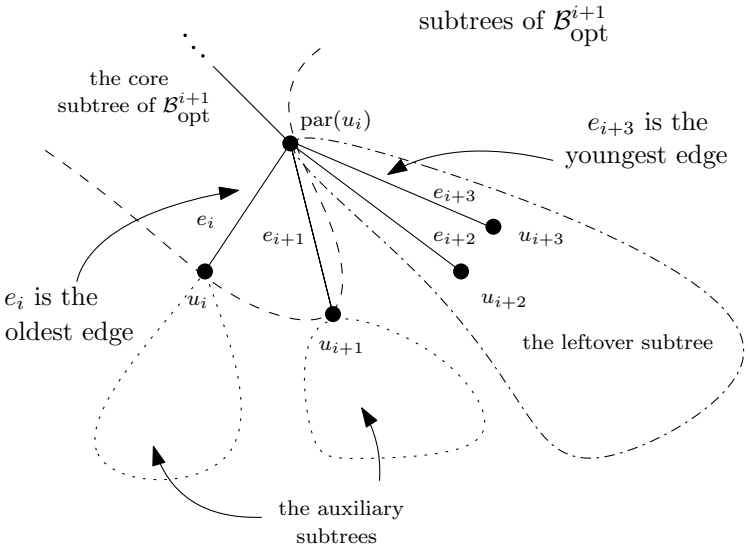


Figure 5.2: An illustration of the notation used in this section: u_i and u_{i+1} belong to the set of terminal vertices of $\mathcal{B}^i_{\text{opt}}$.

$\uparrow_e(\mathcal{S}) = \downarrow_e(\mathcal{S}) = 0$. Let $e_i = \{u_i, \text{par}(u_i)\}$, that is, we identify the index of an edge with the index of its vertex further from the root. For each edge e_i, we define $C_{\text{par}(u_i)}$ to be the set of indices k of the edges e_k directly below $\text{par}(u_i)$.

Denote by R_j, $j \in \{1, \ldots, i\}$, a broadcast strategy for e_j and the auxiliary subtree of u_j. The construction of R_j depends on which one of the six Cases defined at the beginning of this subsection is e_j.

Consider the construction of R_j when e_j is of Case 3. Step 2 on e_j is followed in R_j by all actions in the auxiliary subtree of u_j. These actions are the same as those performed by $\mathcal{B}^0_{\text{opt}}$ in T_{u_j}. Then, the same number of upwards traversals on e_j as in $\mathcal{B}^0_{\text{opt}}$ is made in Step 3 in R_j.

Consider the construction of R_j when e_j is of either Case 1 or 2. In these Cases, Step 1 on e_i occurs in $\mathcal{B}^0_{\text{opt}}$ and this first upward traversal of e_i is omitted in R_i. Recall that the surplus energy is moved to r in the convergecast stage preceding $\mathcal{B}^i_{\text{opt}}$, namely $\mathcal{C}^i_{\text{opt}}$. This is done with no additional cost, by Claim 5.2.1. Then, Step 2 on e_i is made as the first action in R_i. It is followed by Step 1 that occurs in $\mathcal{B}^0_{\text{opt}}$ on all edges in the subtree of T_{u_i}. The agent, denoted as a, which makes the move of Step 1 on e_i in $\mathcal{B}^0_{\text{opt}}$ initially possesses no energy and is at u_i after R_i executes its moves of Step 1 below u_i. An additional energy exchange action between a and the agent making the move of Step 2 on e_i transfers $w(e_i)$ units of energy to a. These actions are then followed by the remaining actions in the subtree T_{u_i}, which are the same as those performed by $\mathcal{B}^0_{\text{opt}}$ in T_{u_i}. Afterwards, the same number of upward moves as in Step 3 in $\mathcal{B}^0_{\text{opt}}$ is made on e_i in R_i. Finally

one more upward traversal of e_i is added, called the *exit move*. This traversal is done by a. Note that at most one agent traverses e_i in Step 1. Informally speaking, in these two Cases the move of Step 1 on e_i is shifted to Step 3 in R_i.

Consider the construction of R_j when e_j is of Case 5 or 6: the same traversals as in $\mathcal{B}^0_{\text{opt}}$ are made on u_j followed by the same actions as those performed by $\mathcal{B}^0_{\text{opt}}$ in T_{u_j}. Note that the assigned order of actions is different than in Case 3.

Consider the construction of R_j when e_j is of Case 4. Step 1 on e_j occurs in $\mathcal{B}^0_{\text{opt}}$ and this upward traversal of e_i is omitted in R_i. Hence, there is one *missing agent* which traversed e_j downwards in $\mathcal{B}^0_{\text{opt}}$, but is not present in R_i. In R_j the agents making Steps 2 and 3 meet on $\text{par}(u_j)$ before any of these moves are made and make energy exchange actions necessary for all of them to traverse e_j downward. Then, one less downward traversal of e_j than in $\mathcal{B}^0_{\text{opt}}$ is made in R_i. Then, these agents wait at u_j for the arrival of the agent, called the *substitute agent*, which would make Step 1 on e_j. Note that the substitute agent is in T_{u_j} in the initial configuration and it arrives to u_j thanks to the same upward traversals of edges of T_{u_j} as those made in Step 1 in T_{u_j} in $\mathcal{B}^0_{\text{opt}}$. From the point of meeting the other agents on u_j, the substitute agent receives the same amount of energy as would be possessed by the missing agent. Then, the agents make the same actions (excluding those aforementioned actions of Step 1 already made) as those performed by $\mathcal{B}^0_{\text{opt}}$ in T_{u_i}, with the substitute agent performing the role of the missing agent.

In order to simplify further references to the constructed R_j their relevant properties are expressed in Lemma 5.4.7. We refer to the i-th property outlined in Lemma 5.4.7 as Lemma 5.4.7.i, e.g. Lemma 5.4.7.1 is the first property.

Lemma 5.4.7. Each R_j has the following properties:

1. $\downarrow_{e_j}(\mathcal{B}^0_{\text{opt}}) - \uparrow_{e_j}(\mathcal{B}^0_{\text{opt}}) = \downarrow_{e_j}(\mathcal{B}^j_{\text{opt}}) - \uparrow_{e_j}(\mathcal{B}^j_{\text{opt}})$,
2. the actions on e_j and the subtree directly below e_j can be performed if the amount of energy delivered in Step 2 on e_j in R_j is not lesser than the amount of energy delivered to this subgraph in $\mathcal{B}^{j-1}_{\text{opt}}$ plus $w(e_j)$ if Step 1 occurs on e_j in $\mathcal{B}^0_{\text{opt}}$,
3. the actions in R_j require at least one and at most $\max\{1, \downarrow_{e_j}(\mathcal{B}^0_{\text{opt}}) - \uparrow_{e_j}(\mathcal{B}^0_{\text{opt}})\}$ agents to traverse e_j downwards,
4. $\text{cost}(R_j)$ is not greater than that required by the actions on e_j and the subtree directly below e_j in $\mathcal{B}^0_{\text{opt}}$,
5. there is no Step 1 on e_j in R_j,
6. if e_j is of Case 1, 2 or 3 (Case 4, 5 or 6), then if there are moves in Step 3 on e_j in R_j, all of them are upwards (downwards, respectively)
7. if e_j is of Case 1, 2 or 3, then all actions in the subtree below e_j happen consecutively after Step 2 on e_j and before the first move of Step 3 on e_j. Furthermore, the actions in the subtree directly below e_j are the same as in $\mathcal{B}^0_{\text{opt}}$,

8. if e_j is of Case 4, 5 or 6, then all actions in the subtree below e_j happen consecutively after the last move of Step 3 on e_j. Furthermore, the actions in the subtree directly below e_j are the same as in $\mathcal{B}^0_{\text{opt}}$.

Proof. First, we consider properties which share simple proofs in all Cases. Proofs of Lemma 5.4.7.5, Lemma 5.4.7.6, Lemma 5.4.7.7 and Lemma 5.4.7.8 follow naturally from the construction of R_j. Recall that in Cases 1, 2 and 3 (4, 5, and 6) if there are moves of Step 3 on e_j in $\mathcal{B}^0_{\text{opt}}$, then they are made upwards (downwards respectively). This is because the direction of moves of Step 3 is copied to R_j from $\mathcal{B}^0_{\text{opt}}$ and the exit move, if it exists, is an upward traversal. Thus, Lemma 5.4.7.6 is satisfied. Because the condition of Lemma 5.4.7.8 (Lemma 5.4.7.7) refers only to Cases 1, 2 and 3 (4, 5, and 6 respectively) it is trivially satisfied in the remaining Cases.

A simple counting argument is sufficient to prove Lemma 5.4.7.1. In Cases 3, 5 and 6 the undertaken actions on e_j are the same in $\mathcal{B}^0_{\text{opt}}$ and R_j, therefore Lemma 5.4.7.1 is trivially satisfied. In Cases 1 and 2 we remove one upward traversal in Step 1 but add one upward traversal in Step 3 (namely the exit move). Thus, $\uparrow_{e_j}(\mathcal{B}^0_{\text{opt}}) = \downarrow_{e_j}(R_j)$ and there is only one downward traversal in both strategies. In Case 4, we remove one upward and add one downward traversal. Thus:

$$\downarrow_{e_j}(\mathcal{B}^j_{\text{opt}}) - \uparrow_{e_j}(\mathcal{B}^j_{\text{opt}}) = \downarrow_{e_j}(\mathcal{B}^0_{\text{opt}}) + 1 - \uparrow_{e_j}(\mathcal{B}^0_{\text{opt}}) - 1.$$

As a corollary to this counting argument, the number of traversals of e_j in R_j is not greater than in $\mathcal{B}^0_{\text{opt}}$. Furthermore, because in each Case the traversals below e_j are the same in both $\mathcal{B}^0_{\text{opt}}$ and R_j, their cost is the same. Thus, Lemma 5.4.7.4 is satisfied.

Next, consider Lemma 5.4.7.3 divided into two parts: Cases 1, 2 and 3 and Cases 4, 5 and 6. By construction, the order of actions in the subtree below e_j is the same relative to each other, but they differ in their placement in $\mathcal{B}^j_{\text{opt}}$ and $\mathcal{B}^0_{\text{opt}}$. Thus, the conditions regarding the initial configuration of agents in Observation 5.4.6 and Observation 5.4.7 are fulfilled. In this section of the proof we assume that the energy is not a concern (i.e. energy requirements in our observations are assumed to be fulfilled), as it will be considered in the proof of Lemma 5.4.7.2. Note that in Cases 1, 2 and 3 $\downarrow_{e_j}(\mathcal{B}^0_{\text{opt}}) - \uparrow_{e_j}(\mathcal{B}^0_{\text{opt}}) \leq 1$, because only one agent traverses e_j downward in $\mathcal{B}^0_{\text{opt}}$. Therefore, only one case, such that actions in R_j can be done with one additional agent present on $\text{par}(u_j)$, needs to be considered. First, this agent traverses e_j downwards with all data packets making Step 2. Then, the moves copied from $\mathcal{B}^0_{\text{opt}}$ are made in the auxiliary subtree of u_j. By Observation 5.4.6, the ability to perform these moves does not depend on the existence of Step 1 on e_j (recall that energy is not considered in this argument). Since the moves in T_{u_j} are the same, the same number of agents from T_{u_j} ends up in R_j on u_j after these moves, as in $\mathcal{B}'_{\text{opt}}$. Thus, these agents can continue to make the same number of moves in Step 3 on e_j in R_j as in $\mathcal{B}^0_{\text{opt}}$. Finally, if there was Step 1 on e_j, then there is one more agent on u_j and one less on $\text{par}(u_j)$ than in $\mathcal{B}^0_{\text{opt}}$, because there is no Step 1 in R_j. As mentioned in the construction, the

additional exit move is done by this agent. Thus, we have completed all moves in R_j.

In Cases 5 and 6 the actions on e_j and the subtree T_{u_j} are the same in both $\mathcal{B}^0_{\text{opt}}$ and R_j. Since in both strategies there are no upward moves $\downarrow_{e_j}(R_j) = \downarrow_{e_j}(\mathcal{B}^0_{\text{opt}})$. Furthermore, in R_j these moves are made before any move in the subtree T_{u_j}. By Observation 5.4.7, this amount of agents is sufficient to make the same moves on T_{u_j} as in $\mathcal{B}^0_{\text{opt}}$. In Case 4, since there is Step 1 on e_j in $\mathcal{B}^0_{\text{opt}}$, Lemma 5.4.7.3 requires that all traversals of e_j are done in R_j by $\downarrow_{e_j}(R_j) = \downarrow_{e_j}(\mathcal{B}^0_{\text{opt}}) - 1$ agents. Hence, there is a risk of there being one less agent on u_j than in $\mathcal{B}^0_{\text{opt}}$. This is addressed by the replacement of the missing agent by the substitute agent, which in turn is done by altering the order of actions in T_{u_j}. By Observation 5.4.5, the substitute agent can arrive at u_j before the arriving agents make further moves of Step 2 on the edges below u_j. Thus, at the end of Step 3 on e_j and Step 1 below e_j in R_j the number of agents on u_j is the same the number of agents which enter T_{u_j} in $\mathcal{B}^0_{\text{opt}}$. This amount of agents is sufficient to make the same moves on T_{u_j} as in $\mathcal{B}^0_{\text{opt}}$.

Finally, consider Lemma 5.4.7.2. We have already considered the amount of agents needed to make the actions in R_j in the proof of Lemma 5.4.7.3, now we focus on the amount of energy required. If e_j is either of Case 3, 5 or 6, then the actions of energy exchange by agents on vertices of e_j are the same in $\mathcal{B}^0_{\text{opt}}$ and R_j, thus Observation 5.4.6 (in Case 3) or Observation 5.4.7 (in Cases 5 and 6) applies. In Case 4 the general argument is analogous, with an additional technical detail. We remark that there is no possibility that the substitute agent held energy needed to make moves of Step 3 on e_j. This is due to the fact that $\mathcal{B}^j_{\text{opt}}$, which R_j is a part of, is preceded by $\mathcal{C}^j_{\text{opt}}$. In $\mathcal{C}^j_{\text{opt}}$ the energy that would be used to make the move of Step 1 on e_j is moved to r. In $\mathcal{C}^i_{\text{opt}}$, $i \in \{0, \ldots, j-1\}$ any energy that would be held by the agent after making the move of Step 1 on e_j in $\mathcal{B}^0_{\text{opt}}$ could be used only to make further upward moves of Step 1. Since an energy exchange action on $\text{par}(e_j) \neq r$ between the agents making Step 1 and Steps 2 and 3 on e_j did not happen in $\mathcal{B}^i_{\text{opt}}$, it also is not needed in $\mathcal{B}^j_{\text{opt}}$.

Consider the remaining Cases 1 and 2. In R_j the agent making the exit move initially has no energy. We compensate for this by supplying the agent making the move of Step 2 on e_j with the additional $w(e_j)$ units of energy, which are exchanged with the agent making the exit move. Note that Lemma 5.4.7.2 accounts for this increase. The remaining energy is distributed as in $\mathcal{B}^0_{\text{opt}}$.

Thus, in each Case the amount of energy delivered to the subtree of T_{u_j} is the same as in $\mathcal{B}^0_{\text{opt}}$. The actions in the subtree T_{u_j} are the same in both $\mathcal{B}^0_{\text{opt}}$ and R_j. Hence, by Observation 5.4.6, these actions and the following actions of Step 3 on e_j can be made and Lemma 5.4.7.2 holds. □

We now move towards constructing the broadcast strategy $\mathcal{B}^i_{\text{opt}}$ for each $i \in \{1, \ldots, n-1\}$, which is done in the form of a pseudo-code in Algorithm CORE EXPANSION. This strategy uses R_j as one of its building blocks. Our key claim (Lemma 5.4.8) says that some of the $\mathcal{B}^j_{\text{opt}}$'s are valid broadcast strategies. We

remark that in particular Lemma 5.4.8 will imply that $\mathcal{B}^{n-1}_{\text{opt}}$ is a valid broadcast in which Step 1 does not occur, and thus this is our final desired strategy. We use Lemma 5.4.7 to prove Lemma 5.4.8 by induction. In the proof we use Lemma 5.4.7.8, Lemma 5.4.7.7 and Lemma 5.4.7.5 to describe the structure of the constructed strategy. Then we verify that agents can make these moves, first by counting traversals of edges using Lemma 5.4.7.1, Lemma 5.4.7.6 and Lemma 5.4.7.3 in addition to the aforementioned properties. Finally we address the amount of energy needed to make these moves using Lemma 5.4.7.8, Lemma 5.4.7.7 and Lemma 5.4.7.2.

Algorithm CORE EXPANSION: An algorithm that constructs $\mathcal{B}^{i}_{\text{opt}}$ for which it takes $\mathcal{B}^{0}_{\text{opt}}, \ldots, \mathcal{B}^{i-1}_{\text{opt}}$ as an input.

1 Use the final configuration of $\mathcal{C}^{i}_{\text{opt}}$ as the initial configuration of $\mathcal{B}^{i}_{\text{opt}}$
2 **if** e_i is the oldest **then**
3 A_i = the maximal sequence of actions before the first move in $T_{\text{par}(u_i)}$ in $\mathcal{B}^{i-1}_{\text{opt}}$.
4 **else**
5 $A_i = \mathcal{B}^{i-1}_{\text{opt}}$
6 Compute R_i ▷ `the actions on` e_i `and below` e_i `depending on Cases 1-6`
7 **if** e_i is the youngest **then**
8 Let e_j be the oldest edge directly below $\text{par}(e_i)$
9 M_i = the maximal sequence of actions outside of $T_{\text{par}(u_i)}$ after the last move in $T_{\text{par}(u_i)}$ in $\mathcal{B}^{j-1}_{\text{opt}}$
10 **else**
11 Let M_i be the empty sequence
12 **return** $\mathcal{B}^{i}_{\text{opt}} = A_i \oplus R_i \oplus M_i$

We provide some informal comments regarding selected steps of Algorithm CORE EXPANSION and Figure 5.3 to aid in forming intuitions. Note that, in line 3, u_i is a child of a terminal vertex of $\mathcal{B}^{i-1}_{\text{opt}}$ in $\widetilde{T}_{i-1}$, and hence $T_{\text{par}(u_i)}$ is an auxiliary subtree in $\mathcal{B}^{i-1}_{\text{opt}}$. This is because e_i is the oldest. Let $T'_{\text{par}(u_i)}$ denote the subtree induced by $E(T) \setminus E(T_{\text{par}(u_i)})$. In such case, we claim that $\mathcal{B}^{i-1}_{\text{opt}}$ is a broadcast strategy and we thus trim it by taking A_i, called the *prefix strategy* of $\mathcal{B}^{i}_{\text{opt}}$, on the edges of $T'_{\text{par}(u_i)}$ as the beginning of the sequence of actions in $\mathcal{B}^{i}_{\text{opt}}$. The remaining actions on $T'_{\text{par}(u_i)}$, called the *suffix strategy* of $\mathcal{B}^{i}_{\text{opt}}$, are discarded. See Figure 5.3(a) and (b). If e_i is not the oldest (see line 5), then it follows from the construction of $\mathcal{B}^{i-1}_{\text{opt}}$ that it does not perform any actions on e_k and T_{u_k}, such that e_k is a younger 'sibling' of e_i. Thus, in both cases we have the property that the prefix strategy A_i does not make any actions on e_i and below it. Then, in line 6, we take R_i having the properties listed in Lemma 5.4.7. Recall that R_i is a

strategy that 'works' on e_i and the subtree below this edge. Note that the edge e_j in line 8 is the oldest 'sibling' of e_i. The strategy M_i is then either borrowed from an earlier strategy $\mathcal{B}^{j-1}_{\text{opt}}$ in line 9 or is omitted (M_i being empty) in case when e_i is not the youngest (line 11). As a consequence, in the latter case $\mathcal{B}^{i}_{\text{opt}}$ is a sequence of actions that can be executed, however it is not a broadcast strategy. See Figure 5.3(c). Intuitively, in the former case we claim that $\mathcal{B}^{j-1}_{\text{opt}}$ is a broadcast strategy and performs some actions in the subtree $T'_{\text{par}(u_i)}$ (note that $T_{\text{par}(u_i)}$ is an auxiliary subtree of $\mathcal{B}^{j-1}_{\text{opt}}$). We thus take as the M_i a part of $\mathcal{B}^{j-1}_{\text{opt}}$ on $T'_{\text{par}(u_i)}$, more precisely the aforementioned discarded suffix strategy of $\mathcal{B}^{j}_{\text{opt}}$, with the exclusion of $T_{\text{par}(u_i)}$. See Figure 5.3(d). A formal analysis of these intuitions is in the proof of Lemma 5.4.8.

Lemma 5.4.8. If there is no leftover subtree in $\mathcal{B}^{z}_{\text{opt}}$ ($E(T) \setminus E(\widetilde{T}_z) = \emptyset$) then $\mathcal{B}^{z}_{\text{opt}}$ is a broadcast strategy.

Proof. The proof is by induction. For the base case, take $z = 0$ and $\mathcal{B}^{0}_{\text{opt}}$. There is no leftover subtree of $\mathcal{B}^{0}_{\text{opt}}$ and $\mathcal{B}^{0}_{\text{opt}}$ is a broadcast strategy, thus trivially satisfying the basis.

For the induction step, the leftover subtree of $\mathcal{B}^{z}_{\text{opt}}$ does not exist if e_z is a youngest edge. Thus, choose $I_{\text{par}(u_z)} = \{a, \ldots, z\}$, where e_a is the oldest edge and e_z is the youngest edge directly below $\text{par}(u_z)$. We only analyze the case when $a \neq z$, as the argument is simplified for $a = z$ (this is the case when $i = j$ in line 8, i.e., e_j has no siblings). Note that it is not necessarily true that $\text{par}(u_z) = u_{a-1}$. If $\text{par}(u_z) \neq r$, let $e' = \{\text{par}(u_z), \text{par}(\text{par}(u_z))\}$ (see Figure 5.4 for an example of such case).

By definition, e_a is the oldest, therefore due to the chosen BFS order we can make the following statements about $\mathcal{B}^{a-1}_{\text{opt}}$. Firstly, e_{a-1} is the youngest edge and $\text{par}(u_{a-1}) \neq \text{par}(u_a)$. Next, there is no leftover subtree of $\mathcal{B}^{a-1}_{\text{opt}}$ (see Figure 5.4(a)). This allows us to conclude that, by induction assumption, $\mathcal{B}^{a-1}_{\text{opt}}$ is a broadcast strategy. Finally, $\text{par}(u_a) = \text{par}(u_z)$ is a terminal vertex and $T_{\text{par}(u_z)}$ is the auxiliary subtree of $\text{par}(u_z)$ in $\mathcal{B}^{a-1}_{\text{opt}}$, but they are no longer a terminal vertex and an auxiliary subtree, respectively, of $\mathcal{B}^{a}_{\text{opt}}$ (see Figure 5.4(b)).

We divide $\mathcal{B}^{a-1}_{\text{opt}}$ into 3 consecutive sequences of actions: the sequence of actions before the first move in $T_{\text{par}(u_z)}$, the sequence of actions in $T_{\text{par}(u_z)}$ (an auxiliary subtree) and the sequence of actions after the last move in $T_{\text{par}(u_z)}$. This can be done because of the following. Consider e' being of Case 1, 2 or 3, thus Lemma 5.4.7.7 is relevant. By Lemma 5.4.7.5, the first action on each edge of the core subtree is in Step 2. All actions up to and including Step 2 on e', if present, are done before any action in $T_{\text{par}(u_z)}$. These actions are finished before Step 3 on e', by the order specified in Lemma 5.4.7.7. Furthermore, by the same lemma, the actions in these sequences happen consecutively. If e_{a-1} is of Case 4, 5 or 6, then Lemma 5.4.7.8 is relevant and an analogous argument can be made. These are the only possibilities. Denote the sequence of actions in $\mathcal{B}^{a-1}_{\text{opt}}$ before (after) the actions in $T_{\text{par}(u_z)}$ as A' (M', respectively) for a future reference. Let R' denote

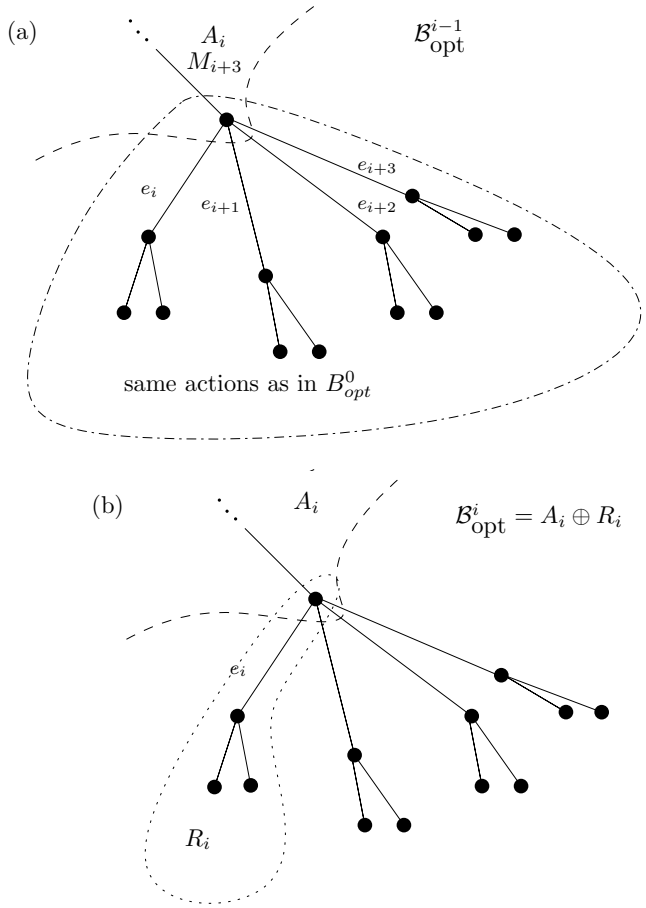


Figure 5.3: An example of execution of Algorithm CORE EXPANSION where we illustrate four subsequent calls producing strategies $\mathcal{B}^{i-1}_{\text{opt}}, \ldots, \mathcal{B}^{i+3}_{\text{opt}}$: (a) a subset of actions of $\mathcal{B}^{i-1}_{\text{opt}}$ will be chosen as A_i in $\mathcal{B}^i_{\text{opt}}$ (note that the illustration represents this choice, not the building of $\mathcal{B}^{i-1}_{\text{opt}}$, and A_i does not represent every action on $T'_{\text{par}(u_i)}$) , followed by processing edges such that (b) e_i is the oldest, (c) e_{i+1} is neither the oldest nor the youngest, and (d) e_{i+1} is the youngest.

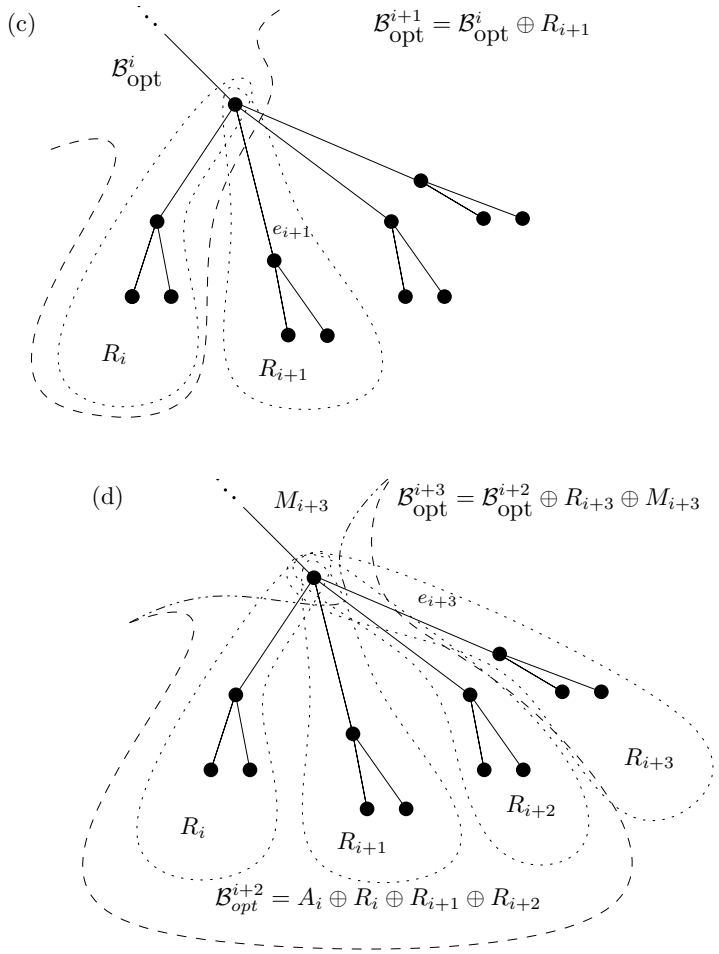


Figure 5.3: An example of execution of Algorithm CORE EXPANSION where we illustrate four subsequent calls producing strategies $\mathcal{B}^{i-1}_{\text{opt}}, \ldots, \mathcal{B}^{i+3}_{\text{opt}}$: (a) a subset of actions of $\mathcal{B}^{i-1}_{\text{opt}}$ will be chosen as A_i in $\mathcal{B}^{i}_{\text{opt}}$ (note that the illustration represents this choice, not the building of $\mathcal{B}^{i-1}_{\text{opt}}$, and A_i does not represent every action on $T'_{\text{par}(u_i)}$) , followed by processing edges such that (b) e_i is the oldest, (c) e_{i+1} is neither the oldest nor the youngest, and (d) e_{i+1} is the youngest.

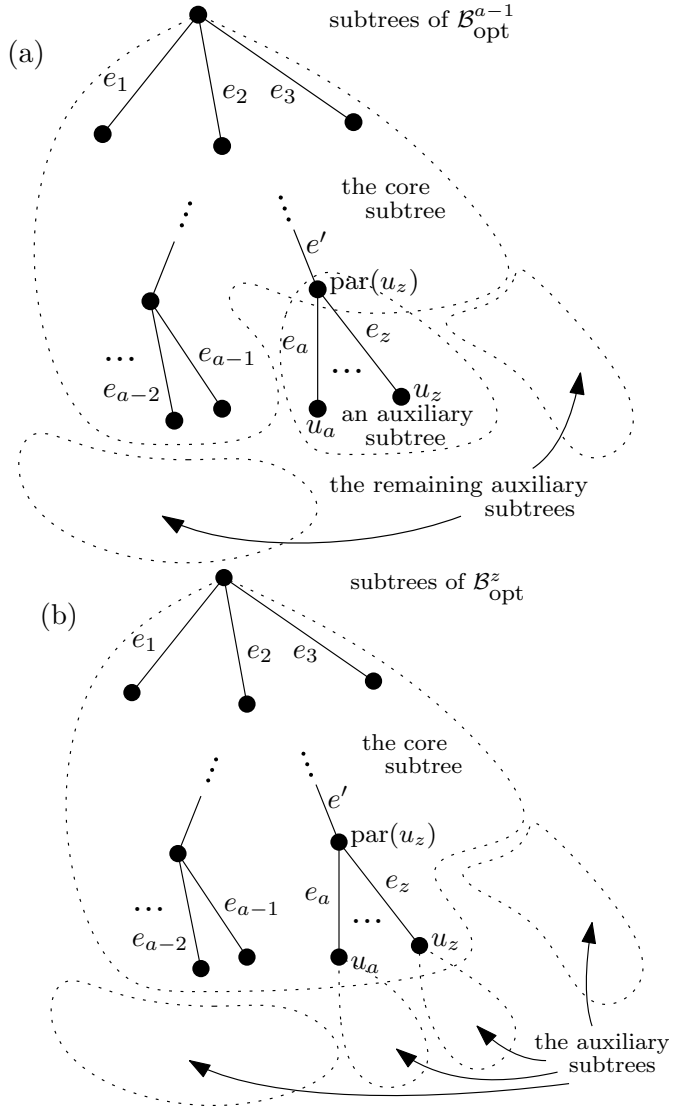


Figure 5.4: An illustration of the notation used in the proof of Lemma 5.4.8: (a) the subtrees of $\mathcal{B}^{a-1}_{\text{opt}}$ and (b) $\mathcal{B}^{z}_{\text{opt}}$

the actions in $T_{\mathrm{par}(u_z)}$ for completeness sake. Given this notation $\mathcal{B}^{a-1}_{\mathrm{opt}}$ can be expressed as:

$$\mathcal{B}^{a-1}_{\mathrm{opt}} = A' \oplus R' \oplus M'. \tag{5.2}$$

The edge e_a is the oldest, so the condition of line 2 of Algorithm CORE EXPANSION is fulfilled during the construction of $\mathcal{B}^{a}_{\mathrm{opt}}$. We assign to A_a (see line 3) the sequence of actions before the first move in $T_{\mathrm{par}(u_z)}$ in $\mathcal{B}^{a-1}_{\mathrm{opt}}$, i.e.

$$A_a = A'. \tag{5.3}$$

By definition of A_a, the actions in A_a can be executed regardless of the actions in R_a. On the other hand we abandon a sequential proof of the capability of execution of R_a in favour of a uniform argument that actions in each R_k, $k \in I_{\mathrm{par}(u_z)}$, (including R_a, because $a \in I_{\mathrm{par}(u_z)}$) can be made. In order to do so we analyze the result of calling the algorithm for each $\mathcal{B}^{k}_{\mathrm{opt}}, k \in I_{\mathrm{par}(u_z)}$. Thus, $\mathcal{B}^{a}_{\mathrm{opt}} = A' \oplus R_a \oplus M_a$ (see line 12), where M_a is an empty sequence because e_a is not the youngest.

Consider e_j, $j \in C$, such that e_j is not the oldest nor the youngest, and thus line 5 applies. In this case M_j is an empty sequence and, by lines 12 and 5:

$$A_j = \mathcal{B}^{j-1}_{\mathrm{opt}} = A_{j-1} \oplus R_{j-1}. \tag{5.4}$$

Thus, by applying (5.4) multiple times,

$$\mathcal{B}^{j}_{\mathrm{opt}} = \mathcal{B}^{j-1}_{\mathrm{opt}} \oplus R_j = A_a \oplus R_a \oplus \cdots \oplus R_{j-1} \oplus R_j. \tag{5.5}$$

Since e_z is the youngest, by lines 5, 7 and 12, and by (5.5) used for $j = z-1$:

$$\mathcal{B}^{z}_{\mathrm{opt}} = A_z \oplus R_z \oplus M_z = \mathcal{B}^{z-1}_{\mathrm{opt}} \oplus R_z \oplus M_z = A_a \oplus R_a \oplus \ldots \oplus R_{z-1} \oplus R_z \oplus M_z. \tag{5.6}$$

Because e_z is the youngest, therefore there is no leftover subtree of $\mathcal{B}^{z}_{\mathrm{opt}}$. Thus, in order to complete the proof we have to demonstrate that $\mathcal{B}^{z}_{\mathrm{opt}}$ in (5.6) is a broadcast strategy.

Notice that, by lines 7-9 of the algorithm, M_z is the maximal sequence of actions outside of $T_{\mathrm{par}(u_z)}$ after the last move in $T_{\mathrm{par}(u_z)}$ in $\mathcal{B}^{a-1}_{\mathrm{opt}}$. Thus, we substitute M' for M_z and apply (5.3) to the rightmost side of (5.6), in order to obtain:

$$\mathcal{B}^{z}_{\mathrm{opt}} = A' \oplus R_a \oplus \ldots \oplus R_{z-1} \oplus R_z \oplus M'. \tag{5.7}$$

We have shown that if actions in each R_k, $k \in I_{\mathrm{par}(u_z)}$, and M' can be made, then $\mathcal{B}^{z}_{\mathrm{opt}}$ is a broadcast strategy.

Let us address a corollary implication of Equation (5.7) before we continue. Note that, since $R_a \oplus \ldots \oplus R_{z-1} \oplus R_z$ does not make actions in the core subtree of $\mathcal{B}^{a-1}_{\mathrm{opt}}$, the same traversals are performed on each edge of this subtree in $\mathcal{B}^{a-1}_{\mathrm{opt}}$ and $\mathcal{B}^{z}_{\mathrm{opt}}$. Since an equation analogous to (5.7) can be written for any broadcast strategy among the strategies $\{\mathcal{B}^{0}_{\mathrm{opt}}, \ldots, \mathcal{B}^{a-1}_{\mathrm{opt}}\}$, we obtain that the actions in the core subtree of an arbitrary $\mathcal{B}^{i}_{\mathrm{opt}}$, $i \in \{0, \ldots, a-1\}$ are the same in any $\mathcal{B}^{l}_{\mathrm{opt}}$, $l \geq i$,

provided that in $\mathcal{B}^l_{\text{opt}}$ R_i is not an empty sequence. Furthermore, by the chosen BFS order, the actions on the edge e_i were introduced in R_i. By Lemma 5.4.7.1,

$$\downarrow_{e_i}(\mathcal{B}^{a-1}_{\text{opt}}) - \uparrow_{e_i}(\mathcal{B}^{a-1}_{\text{opt}}) = \downarrow_{e_i}(\mathcal{B}^{i}_{\text{opt}}) - \uparrow_{e_i}(\mathcal{B}^{i}_{\text{opt}}) = \downarrow_{e_i}(\mathcal{B}^{0}_{\text{opt}}) - \uparrow_{e_i}(\mathcal{B}^{0}_{\text{opt}}) \tag{5.8}$$

for an arbitrary edge e_i of the core subtree of $\mathcal{B}^{a-1}_{\text{opt}}$. We will recall this fact later.

Now we verify that actions in each R_k, $k \in I_{\text{par}(u_z)}$, can be made, first by checking that number of agents required by Lemma 5.4.7.3 is available, then whether the amount of energy required in Lemma 5.4.7.2 is delivered to all agents making moves in R_k. Furthermore we make sure that after completion of the actions in R_k, $k \in I_{\text{par}(u_z)}$, enough agents remain to perform the actions in M'.

We will show that actions in each R_k, $k \in I_{\text{par}(u_z)}$, can be made assuming that energy is not a concern, i.e. we argue that for each edge traversal the corresponding agent is available to make the traversal. Note that line 1 of Algorithm CORE EXPANSION changes the initial configuration of each $\mathcal{B}^i_{\text{opt}}, i \in I_{\text{par}(u_z)}$, however the positions of the agents remain the same because, by definition of $\mathcal{C}^i_{\text{opt}}$, $\mathcal{C}^i_{\text{opt}} \approx \mathcal{C}^0_{\text{opt}}$ for each i. By the fact that each e_k, $k \in I_{\text{par}(u_z)}$, is directly below the vertex $\text{par}(u_z)$, which belongs to the core subtree of both $\mathcal{B}^{a-1}_{\text{opt}}$ and $\mathcal{B}^z_{\text{opt}}$, we can keep track of the number of agents available in $T_{\text{par}(u_z)}$ to verify that the actions in every R_k can be made.

First, consider how actions on the subtree induced by $E(T) \setminus E(T_{\text{par}(u_z)})$ influence the number of agents at $\text{par}(u_z)$. If $\text{par}(u_z) = r$, then $\mathcal{B}^{a-1}_{\text{opt}} = \mathcal{B}^0_{\text{opt}}$ and trivially there are no actions on $T_r = T$. Hence, in the following analysis we assume that $\text{par}(u_z) \neq r$. Note that, by Lemma 5.4.7.5, there are no upward traversals of $e' = \{\text{par}(u_z), \text{par}(\text{par}(u_z))\}$ before the move of Step 2 on e'. Consider e' of Case either 1, 2 or 3. There are no downward traversals of e' in Step 3 by Lemma 5.4.7.6. Lemma 5.4.7.7 implies that no upward traversal of e' happens before any action in $T_{\text{par}(u_z)}$. Furthermore, all downward traversals of e' (namely those in Step 2) happen before any action in $T_{\text{par}(u_z)}$. Consider e' of Case either 4, 5 or 6. There are no upward traversals of e' in Step 3 by Lemma 5.4.7.6. Lemma 5.4.7.8 implies that all downward traversals of e' were done before any action in $T_{\text{par}(u_z)}$. Note that in this case there are no upward traversals of e'. The conclusion unifying all of the considered cases is expressed as follows. In every case all actions in $T_{\text{par}(u_z)}$ happen after all resources have been delivered to $\text{par}(u_z)$ via e' in A' and before any have left $\text{par}(u_z)$ in M'. As a corollary, no upward traversals of e' happen in A' and no downward traversals of e' happen in M'. Note that by definitions of the prefix and auxiliary subtrees combined with the chosen BFS order, R_j does not change the number nor order of traversals in any core subtree of $\mathcal{B}^i_{\text{opt}}$ such that $i < j$. Because $A_a = A'$ due to (5.3), the number of agents which arrived at $\text{par}(u_z)$ via e' in $\mathcal{B}^z_{\text{opt}}$ is:

$$\downarrow_{e'}(\mathcal{B}^z_{\text{opt}}) = \downarrow_{e'}(\mathcal{B}^{a-1}_{\text{opt}}). \tag{5.9}$$

Because $M_z = M'$, the number of agents which departed from $\text{par}(u_z)$ via e' is:

$$\uparrow_{e'}(\mathcal{B}^z_{\text{opt}}) = \uparrow_{e'}(\mathcal{B}^{a-1}_{\text{opt}}). \tag{5.10}$$

Note that if $\mathrm{par}(u_z) = r$, then e' does not belong to T, and therefore $\uparrow_{e'}(\mathcal{S}) = \downarrow_{e'}(\mathcal{S}) = 0$, which is consistent with the opening remarks about this case.

Next, we consider the influence of R_k, $k \in I_{\mathrm{par}(u_z)}$, on the number of agents at $\mathrm{par}(u_z)$. Consider only edges in $I_{\mathrm{par}(u_z)}$. Let X be the set of indices of these edges of Cases 1, 2 and 3. Let Y be the set of indices of these edges of Cases 4, 5 and 6. Naturally $I_{\mathrm{par}(u_z)} = X \cup Y$. By Lemma 5.4.7.8 and Lemma 5.4.7.7, the actions on e_k, $k \in I_{\mathrm{par}(u_z)}$, are done sequentially. By the chosen BFS order, the actions on edges of Cases 1, 2 or 3 are done before any action on edge of Case 4, 5 or 6, i.e., $x < y$ for each $x \in X$ and $y \in Y$. By Lemma 5.4.7.7 and Lemma 5.4.7.6, the actions on the edges of Cases 1, 2 or 3 do not reduce the number of agents on $\mathrm{par}(u_z)$. These actions contain one downward traversal, namely the one in Step 2, and at least one upward traversal, thus, by Lemma 5.4.7.3, they require one agent on $\mathrm{par}(u_z)$. At least one agent arrived at $\mathrm{par}(u_z)$ in Step 2 of e' or, if $\mathrm{par}(u_z) = r$, it was present on r by the completion of convergecast beforehand. Therefore, the actions in all R_x, where $x \in X$, are possible.

Consider an arbitrary edge e_y, $y \in Y$. By Lemma 5.4.7.6, in Cases 4, 5 and 6 all traversals are downward, thus with each action on e_y performed in R_y the number of agents on $\mathrm{par}(u_z)$ is reduced. By $x < y$ for each $x \in X$, all actions which deliver agents to $\mathrm{par}(u_z)$ have already happened. Recall that because $M_z = M'$, the agents only depart from $\mathrm{par}(u_z)$ via e' in M_z. Thus, the number of agents on $\mathrm{par}(u_z)$ will not increase with any action following the last action in any R_x, such that $x \in X$. In order to see whether the actions in each R_y, $y \in Y$, and M' can be made, a counting argument is sufficient. Namely we have to disprove the possibility that the number of agents on $\mathrm{par}(u_z)$ at the end of $\mathcal{B}^z_{\mathrm{opt}}$ is negative, which would imply that an impossible action was made. This is because, if for an arbitrary R_y, $y \in Y$, the number of agents on $\mathrm{par}(u_z)$ would be insufficient (i.e. negative, after actions in R_y were made), then this number stays negative until the end of $\mathcal{B}^z_{\mathrm{opt}}$. We do this by comparison with the number of agents in $\mathrm{par}(u_z)$ in $\mathcal{B}^0_{\mathrm{opt}}$, which we know is a broadcast strategy.

Let $s_{\mathrm{par}(u_z)}(\mathcal{S})$ be the initial number of agents on $\mathrm{par}(u_z)$ in a strategy $\mathcal{S}$. Note that $s_{\mathrm{par}(u_z)}(\mathcal{B}^0_{\mathrm{opt}}) = s_{\mathrm{par}(u_z)}(\mathcal{B}^z_{\mathrm{opt}})$. Let $a_{\mathrm{par}(u_z)}(\mathcal{S})$ be the final number of agents on $\mathrm{par}(u_z)$ in a strategy $\mathcal{S}$. From now on we assume that enough energy and agents were delivered to each R_k, $k \in I_{\mathrm{par}(u_z)}$, and see whether performed actions lead to $a_{\mathrm{par}(u_z)}(\mathcal{B}^z_{\mathrm{opt}}) < 0$. For any strategy $\mathcal{S}$ the number of agents on a vertex v is the initial number of agents on v plus the difference between the number of traversals to v and from v. In particular for $\mathrm{par}(u_z)$ the formula is:

$$a_{\mathrm{par}(u_z)}(\mathcal{S}) = s_{\mathrm{par}(u_z)}(\mathcal{S}) + \downarrow_{e'}(\mathcal{S}) + \sum_{k \in I_{\mathrm{par}(u_z)}} \big(\uparrow_{e_k}(\mathcal{S}) - \downarrow_{e_k}(\mathcal{S})\big) - \uparrow_{e'}(\mathcal{S}).$$

Therefore:

$$a_{\mathrm{par}(u_z)}(\mathcal{B}^z_{\mathrm{opt}}) = s_{\mathrm{par}(u_z)}(\mathcal{B}^z_{\mathrm{opt}}) + \downarrow_{e'}(\mathcal{B}^z_{\mathrm{opt}}) + \sum_{k \in I_{\mathrm{par}(u_z)}} \big(\uparrow_{e_k}(\mathcal{B}^z_{\mathrm{opt}}) - \downarrow_{e_k}(\mathcal{B}^z_{\mathrm{opt}})\big) - \uparrow_{e'}(\mathcal{B}^z_{\mathrm{opt}}).$$

By Equations (5.9) and (5.10):

$$a_{\mathrm{par}(u_z)}(\mathcal{B}^z_{\mathrm{opt}}) = s_{\mathrm{par}(u_z)}(\mathcal{B}^z_{\mathrm{opt}}) + \downarrow_{e'}(\mathcal{B}^{a-1}_{\mathrm{opt}}) + \sum_{k \in I_{\mathrm{par}(u_z)}} \left(\uparrow_{e_k}(\mathcal{B}^z_{\mathrm{opt}}) - \downarrow_{e_k}(\mathcal{B}^z_{\mathrm{opt}})\right) - \uparrow_{e'}(\mathcal{B}^{a-1}_{\mathrm{opt}}).$$

Consider the traversals of e_k for $k \in I_{\mathrm{par}(u_z)}$ in $\mathcal{B}^z_{\mathrm{opt}}$. Note that every u_k, $k \in I_{\mathrm{par}(u_z)}$, is a terminal vertex in $\mathcal{B}^z_{\mathrm{opt}}$ and, by assumption, the actions in each R_k are made. Since these strategies perform actions in disjoint subtrees, by Equation (5.6), we obtain $\uparrow_{e_k}(\mathcal{B}^k_{\mathrm{opt}}) - \downarrow_{e_k}(\mathcal{B}^k_{\mathrm{opt}}) = \uparrow_{e_k}(\mathcal{B}^z_{\mathrm{opt}}) - \downarrow_{e_k}(\mathcal{B}^z_{\mathrm{opt}})$ for each $k \in I_{\mathrm{par}(u_z)}$. Furthermore, by Lemma 5.4.7.1, $\uparrow_{e_k}(\mathcal{B}^k_{\mathrm{opt}}) - \downarrow_{e_k}(\mathcal{B}^k_{\mathrm{opt}}) = \uparrow_{e_k}(\mathcal{B}^0_{\mathrm{opt}}) - \downarrow_{e_k}(\mathcal{B}^0_{\mathrm{opt}})$. Thus:

$$a_{\mathrm{par}(u_z)}(\mathcal{B}^z_{\mathrm{opt}}) = s_{\mathrm{par}(u_z)}(\mathcal{B}^z_{\mathrm{opt}}) + \downarrow_{e'}(\mathcal{B}^{a-1}_{\mathrm{opt}}) + \sum_{k \in I_{\mathrm{par}(u_z)}} \left(\uparrow_{e_k}(\mathcal{B}^0_{\mathrm{opt}}) - \downarrow_{e_k}(\mathcal{B}^0_{\mathrm{opt}})\right) - \uparrow_{e'}(\mathcal{B}^{a-1}_{\mathrm{opt}}).$$

Consider the traversals of e' in $\mathcal{B}^{a-1}_{\mathrm{opt}}$. By (5.8) we obtain:

$$\downarrow_{e'}(\mathcal{B}^{a-1}_{\mathrm{opt}}) - \uparrow_{e'}(\mathcal{B}^{a-1}_{\mathrm{opt}}) = \downarrow_{e'}(\mathcal{B}^0_{\mathrm{opt}}) - \uparrow_{e'}(\mathcal{B}^0_{\mathrm{opt}}). \tag{5.11}$$

Therefore:

$$a_{\mathrm{par}(u_z)}(\mathcal{B}^z_{\mathrm{opt}}) = s_{\mathrm{par}(u_z)}(\mathcal{B}^0_{\mathrm{opt}}) + \downarrow_{e'}(\mathcal{B}^0_{\mathrm{opt}}) + \sum_{k \in I_{\mathrm{par}(u_z)}} \left(\uparrow_{e_k}(\mathcal{B}^0_{\mathrm{opt}}) - \downarrow_{e_k}(\mathcal{B}^0_{\mathrm{opt}})\right) - \uparrow_{e'}(\mathcal{B}^0_{\mathrm{opt}}) = a_{\mathrm{par}(u_z)}(\mathcal{B}^0_{\mathrm{opt}}).$$

The number of agents at $\mathrm{par}(u_z)$ is the same in $\mathcal{B}^0_{\mathrm{opt}}$ and $\mathcal{B}^z_{\mathrm{opt}}$, thus it cannot be negative in $\mathcal{B}^z_{\mathrm{opt}}$. We have proven that all actions in each R_k, $k \in I_{\mathrm{par}(u_z)}$, can be made provided that the amount of energy delivered to $T_{\mathrm{par}(u_z)}$ and each R_k, $k \in I_{\mathrm{par}(u_z)}$, is sufficient.

Let us compare the amount of energy available and needed to make actions in $T_{\mathrm{par}(u_z)}$ in $\mathcal{B}^{a-1}_{\mathrm{opt}}$ and $\mathcal{B}^z_{\mathrm{opt}}$. Because $T_{\mathrm{par}(u_z)}$ is one of the auxiliary subtrees of $\mathcal{B}^{a-1}_{\mathrm{opt}}$, both Lemma 5.4.7.8 and Lemma 5.4.7.7 state that it contains the same actions as $\mathcal{B}^0_{\mathrm{opt}}$ regardless of Case of e'. Therefore, the cost of these actions is the same in both $\mathcal{B}^0_{\mathrm{opt}}$ and $\mathcal{B}^{a-1}_{\mathrm{opt}}$. Thus, by Lemma 5.4.7.4, the amount of energy on $\mathrm{par}(u_z)$ after the actions in $A_a = A'$ is sufficient to perform the actions in $T_{\mathrm{par}(u_z)}$ and follow them up with M' in $\mathcal{B}^{a-1}_{\mathrm{opt}}$.

Now consider $\mathcal{B}^z_{\mathrm{opt}}$. By Equation (5.7), A' (respectively M') is the sequence of actions before (after, respectively) any move in R_k, $k \in I_{\mathrm{par}(u_z)}$, and these actions cost the same amount of energy both in $\mathcal{B}^{a-1}_{\mathrm{opt}}$ and $\mathcal{B}^z_{\mathrm{opt}}$. Let $\overline{T}[e_k]$, $k \in I_{\mathrm{par}(u_z)}$, denote the edge e_k and the auxiliary subtree below u_k. By its definition,

R_k, $k \in I_{\mathrm{par}(u_z)}$, is a sequence of actions on $E(\overline{T}[e_k])$. By Lemma 5.4.7.4 and $\bigcup_{k \in I_{\mathrm{par}(u_z)}} E(\overline{T}[e_k]) = E(T_{\mathrm{par}(u_z)})$, the cost of actions in $T_{\mathrm{par}(u_z)}$ is not greater in $\mathcal{B}^z_{\mathrm{opt}}$ than in $\mathcal{B}^{a-1}_{\mathrm{opt}}$. If the amount of energy on $\mathrm{par}(u_z)$ after the actions in $A' \oplus R_a \oplus \ldots \oplus R_z$ is sufficient to perform all following actions in $\mathcal{B}^{a-1}_{\mathrm{opt}}$, then it is also sufficient in $\mathcal{B}^z_{\mathrm{opt}}$, because $M' = M_z$. Furthermore, the overall cost of $\mathcal{B}^z_{\mathrm{opt}}$ is not greater than that of $\mathcal{B}^{a-1}_{\mathrm{opt}}$.

What remains to be show is that the energy can be distributed properly to make each $R_j, a \leq j \leq z$. Consider line 1 and an arbitrary index j, $a \leq j \leq z$. Recall the definitions of $\mathcal{C}^0_{\mathrm{opt}}$, given in Lemma 5.4.6, and $\mathcal{C}^j_{\mathrm{opt}}$. It follows, that in the final configuration of either of them the amount of energy at r is equal to energy(T) minus the cost of actions in Step 1 in $\mathcal{B}^j_{\mathrm{opt}}$. The final configuration of $\mathcal{C}^j_{\mathrm{opt}}$ is the same as the initial configuration of $\mathcal{B}^j_{\mathrm{opt}}$ for each $j \leq z$. For the sake of consistency we use the latter denomination. By Lemma 5.4.7.5, there is no Step 1 in the core subtree of $\mathcal{B}^{j-1}_{\mathrm{opt}}$. The difference between the amount of energy at r in the initial configuration of $\mathcal{B}^j_{\mathrm{opt}}$ and $\mathcal{B}^{j-1}_{\mathrm{opt}}$ is always non-negative, let it be x. More specifically x is equal to 0 if there is no Step 1 on e_j in $\mathcal{B}^{j-1}_{\mathrm{opt}}$, or $w(e_j)$ if there is Step 1 on e_j in $\mathcal{B}^{j-1}_{\mathrm{opt}}$. Thus, if $x = 0$, then the same amount of energy to e_j is delivered in $\mathcal{B}^j_{\mathrm{opt}}$ as in $\mathcal{B}^{j-1}_{\mathrm{opt}}$, thereby satisfying Lemma 5.4.7.2. On the other hand, if $x = w(e_j)$, then deliver the same amount of energy to e_j plus $w(e_j)$, again satisfying Lemma 5.4.7.2. Thus, by induction, the actions in each $R_j, a \leq j \leq z$, can be made. This finishes the proof. □

Consider the r-gossiping strategy $\mathcal{G}^{n-1}_{\mathrm{opt}} = \mathcal{C}^{n-1}_{\mathrm{opt}} \oplus \mathcal{B}^{n-1}_{\mathrm{opt}}$. We are ready to use Lemma 5.4.5 in order to argue that $\mathrm{cost}(\mathcal{G}^{n-1}_{\mathrm{opt}}) \leq \mathrm{cost}(\mathcal{G}_{\mathrm{opt}})$. Since the whole T is the core subtree of $\mathcal{B}^{n-1}_{\mathrm{opt}}$, $\mathcal{B}^{n-1}_{\mathrm{opt}}$ has no moves of Step 1. Hence (b) of Lemma 5.4.5 is satisfied. Furthermore, all energy is at the root in $\mathcal{C}^{n-1}_{\mathrm{opt}}$ and $\mathcal{C}^{n-1}_{\mathrm{opt}} \approx \mathcal{C}_{\min}$, hence $\mathcal{C}^{n-1}_{\mathrm{opt}}$ equals $\mathcal{C}_{\mathrm{alg}}$ (by Corollary 5.4.1). This trivially satisfies conditions (d) and (c) of Lemma 5.4.5. Recall that $\mathcal{G}'_{\mathrm{opt}} = \mathcal{C}'_{\mathrm{opt}} \oplus \mathcal{B}'_{\mathrm{opt}}$ such that $\mathrm{cost}(\mathcal{G}'_{\mathrm{opt}}) = \mathrm{cost}(\mathcal{G}_{\mathrm{opt}})$ and $\mathcal{C}'_{\mathrm{opt}} \approx \mathcal{C}_{\min}$. Finally, the broadcast stage satisfies $\mathrm{cost}(\mathcal{B}'_{\mathrm{opt}}) \geq \mathrm{cost}(\mathcal{B}^{n-1}_{\mathrm{opt}})$. Since $\mathrm{cost}(\mathcal{B}'_{\mathrm{opt}})$ is assumed to be minimized, then (a) also holds. Recall that, by its definition, $\mathcal{G}_{\mathrm{alg}}$ is an r-gossiping strategy of minimum cost such that $\mathcal{G}_{\mathrm{alg}} = \mathcal{C}_{\min} \oplus \mathcal{B}_{\mathrm{alg}}$ for some broadcast stage $\mathcal{B}_{\mathrm{alg}}$.

Thus all conditions of Lemma 5.4.5 hold and we have proved the following lemma:

Lemma 5.4.9. It holds $\mathrm{cost}(\mathcal{G}_{\mathrm{alg}}) = \mathrm{cost}(\mathcal{G}^{n-1}_{\mathrm{opt}}) \leq \mathrm{cost}(\mathcal{G}_{\mathrm{opt}})$.

Lemma 5.4.9 implies the following.

Lemma 5.4.10. For a given tree T rooted at r, if there exists an r-gossiping strategy, then there exists a structured r-gossiping strategy of minimum cost such that its convergecast stage is $\mathcal{C}_{\min}$. □

Having this lemma, we are ready to finish the proof of our main result.

Proof of Theorem 7. For each $r \in V(T)$, compute the following structured r-gossiping strategy. First, run Algorithm CONVERGECAST to obtain $\mathcal{C}_{\min}$ in time $O(n)$; see Lemma 5.4.3. Then, use the algorithm from Theorem 8 to find a minimum-cost broadcast for the configuration of agents at the end of $\mathcal{C}_{\min}$. This can be done in time $O(k^2 n)$. Thus, the n runs take $O(k^2 n^2)$ time in total. The definition of structured gossiping strategy and Lemma 5.4.10 imply that one choice of r provides a solution to the gossiping problem for the input tree T. □

5.5 Summary and Open Problems

We have proved that gossiping can be solved in polynomial time for trees, several natural open problems remain. The first one refers to the complexity: our algorithm takes $O(k^2 n^2)$ time which is determined by the necessity of n executions of the most expensive subroutine — the broadcast algorithm from [68]. Can this be avoided by e.g. narrowing down the number of potential nodes r or by developing an algorithm that does not decompose gossiping into the two stages treated independently?

One may consider some generalizations of gossiping for trees, where a natural one is when initially the data packets are placed only at selected nodes of the tree, and need to be delivered to all nodes. This problem has properties which shows some similarities to ours, i.e., it can still be partitioned into a convergecast stage and a broadcast stage. However, the difference lies in a potential behaviour of an agent that is initially located at a node v such that there is no data packet initially in T_v. This is because this agent does not necessarily need to move in the convergecast stage and thus it may remain idle to be used in the broadcast stage. In other words, we lose the property that at the mid-point all energy is at the root. We leave the analysis of this problem variation as an open research question.

Other open problems refer to general graphs. We note that the problems are NP-complete in general graphs. For broadcast this follows from a straightforward reduction from a Hamiltonian path problem (here we consider the version of the Hamiltonian path with a constraint that the path needs to start at a given node v of the input simple graph G). The corresponding input to broadcast is then the graph G (with unit edge-weights) and one agent placed at v with the initial energy level $n-1$, where n is the number of nodes in G. In such setting, any successful broadcast from v requires visiting each node exactly once during the $n-1$ edge traversals, which precisely dictates a Hamilton path.

A similar reduction follows for convergecast: add an universal node u to G and an additional pending node u' to u. Again, an agent starting at v and having the initial energy level equal to $n+1$ and performing convergecast needs to visit each node exactly once and moreover, the two lastly visited nodes are u and u' in this order. The latter allows us again to conclude, that this agent's route that delivers all data packets to u', when restricted to G is the required path.

Finally, NP-completeness for gossiping follows similarly (we skip the details) by considering the above extended graph and adding an edge $\{v, u'\}$. Knowing

this complexity status, it is interesting to consider some approximate solution. Typically, for energy-constraint agents two approaches are adopted depending on which parameter of the input instance is relaxed when finding approximations. One is adding more agents, see e.g. [99], and another is scaling the initial energy levels of agents, see e.g. [76].

In the context of mobile agent computing, one may ask what possible restrictions imposed on agents in distributed computations are essential from the point of view of problem solvability. For example, how much more resources (in terms of total energy and/or the number of agents) are needed when only local communication is assumed or when the agents have more limited initial knowledge (e.g. related to the structure of the underlying graph). (Note that our centralized setting is equivalent to the mobile agent system in which a global communication with size-unrestricted messages is used.)

Chapter 6

Conclusions

In lieu of repetition of the discussions in Sections 4.7 and 5.5, we would like to offer the reader only a list of the main results and a compilation of applications of our work.In this work we have proved the following:

1. The heterogeneous graph searching problem is not monotone (Theorem 1) and NP-Hard in the class of trees (Theorem 3).

2. The monotone heterogeneous graph searching problem is NP-Complete in the class of trees (Theorem 4).

3. The (connected) heterogeneous graph searching problem admits a polynomial time algorithm when edges associated with each label form a connected subtree (Theorem 5).

4. The gossiping problem in trees with energy-constrained agents can be decomposed into broadcast and convergcast subproblems (Theorem 6).

5. The gossiping problem admits an $O(k^2n^2)$ time algorithm, where k is the number of agents and n the number of nodes in a tree (Theorem 7).

6.1 Applications

Out of the concern for more practically oriented readers, we have gathered examples of practical, or at least potential, uses of aspects of our work. While not specifically a part of our original research, the survey part of this thesis is also concerned with distributed environments. To this end, connected and internal search strategies better reflect the nature of autonomous entities operating with a hostile entity in mind, as remarked in [11, 13]. Many different variants (both distributed and centralized) were introduced since the inception of the problem, and while not all of them are applicable to physical entities, they could be used by software agents (one such example is cloning [119]). We note that the results

from the discipline can also be adapted to the more human-friendly, 2-dimensional space. Although humans themselves are unlikely to act according to graph searching algorithms, they find their use in the field of robotics. We support this claim with the following examples: [148, 147] (see also other works of Hollinger at al.) and a survey [55].

Our heterogeneous approach fits this broad category of adaptations by allowing to differentiate agents in order to match the needs of a given environment. We trust that it is not a controversial statement to say that the concept of using heterogeneous agents has garnered relatively little attention in theoretical research. Certainly, even in its broadest sense, it is not as widespread as constructions of weighted versions of various graph problems. Nonetheless, we give an example of [179] which models traffic flow with different vehicles assigned to heterogeneous classes and confronts them with real-world scenarios.

Another way to look at the usefulness of graph searching is to connect the results with other recognized graph parameters. In particular, pathwidth and treewidth come to mind. We mention a few commonly cited cross-discipline applications: graph drawing [96], design of algorithms [23] and electric circuits [108, 81], a theory in natural language processing [170], database management [142]. Nevertheless, sometimes a game-like approach is more convenient, e.g. ensuring privacy in distributed systems [133]. Finding these links is not trivial. The authors of [10] note that (their attention was directed to the fact that) connected search number of trees can be used to define Horton–Strahler number, a well established measure in hydrology [150, 218, 219] (first relevant citation is from 1945, over half a century later [139] has been published on the topic). Through this link, tree searching can be related to: memory used in arithmetic calculations [110], mathematical description of a respiratory system [41, 149] and community analysis in social networks [4].

As far as our work on the topic of graph searching is concerned, we hope to provide clues that will (eventually) lead to closing of the open question of NP-Completeness of connected graph searching. [10, 29]. To this end, we follow the route of [29] and provide a model which is non-monotone for trees — a historically promising class to research. Furthermore, our problem admits a relatively simple way of generating a significant number of recontaminations. It remains to be seen if a non-polynomial size strategy (i.e. a certificate that cannot be evaluated in a polynomial time) can be found.

Next, let us discuss gossiping and adjacent problems. As a general all-to-all communication problem [145, 151] it is applicable in: construction of routing tables [210], radio communication [135, 134] (see also works of Gąsieniec at al.), modeling telephone communication [132] and wireless mobile ad-hoc networks [78]. The problem is naturally related to broadcast, one-to-all communication ([52, 63]) and convergcast.

Beyond these links to these common communication problems (classically utilizing messages), the mobile aspect of our work lends itself to a description of a more mundane model of vehicle routing [227]. The angle of optimization for the used amount of energy can be found in creating patterns from mobile entities [221],

load balancing [7], piecemeal graph exploration [6] and the premise of [1], which proposes it as a third general optimization parameter to join the ever popular space and time.

6.2 Contributions

Let us finish with the outline of the contribution of the author of this thesis to the obtained results. The author hereby lays claim to the following:

- Active participation in the discussions leading to facts and lemmas in Section 4.2.2 culminating in the development of ideas behind 4.3 and 4.6. Writing down and developing Section 4.3.
- Refinement of a jointly developed idea of reduction in Section 4.4 into a formal, written proof.
- The extension of the above to a non-monotone model and the associated proof in Section 4.5, including the development of the notation and writing.
- The ideas presented in Section 5.3 are a result of a joint discussion. Nonetheless, Sections 5.4.3, 5.4.2 and in 5.4.4 containing the formalized proof of correctness of the developed approach were written by the author of this thesis. In particular we lay claim to the leading role in the development of the proof in Section 5.4.4.

Bibliography

[1] Susanne Albers. Energy-efficient algorithms. *Communications of the ACM*, 53(5):86–96, 2010.

[2] Brian Alspach, Danny Dyer, Denis Hanson, and Boting Yang. Time constrained graph searching. *Theoretical Computer Science*, 399(3):158–168, 2008.

[3] Julian Anaya, Jérémie Chalopin, Jurek Czyzowicz, Arnaud Labourel, Andrzej Pelc, and Yann Vaxès. Collecting information by power-aware mobile agents. In *Distributed Computing: 26th International Symposium, DISC 2012, Salvador, Brazil, October 16-18, 2012. Proceedings 26*, pages 46–60. Springer, 2012.

[4] Alex Arenas, Leon Danon, Albert Diaz-Guilera, Pablo M Gleiser, and Roger Guimera. Community analysis in social networks. *The European Physical Journal B*, 38:373–380, 2004.

[5] Stefan Arnborg, Derek G Corneil, and Andrzej Proskurowski. Complexity of finding embeddings in ak-tree. *SIAM Journal on Algebraic Discrete Methods*, 8(2):277–284, 1987.

[6] Baruch Awerbuch, Margrit Betke, Ronald L Rivest, and Mona Singh. Piecemeal graph exploration by a mobile robot. *Information and Computation*, 152(2):155–172, 1999.

[7] Yossi Azar. On-line load balancing. *Online algorithms: the state of the art*, pages 178–195, 2005.

[8] Evangelos Bampas, Shantanu Das, Dariusz Dereniowski, and Christina Karousatou. Collaborative delivery by energy-sharing low-power mobile robots. In *Algorithms for Sensor Systems: 13th International Symposium on Algorithms and Experiments for Wireless Sensor Networks, ALGOSENSORS 2017, Vienna, Austria, September 7-8, 2017, Revised Selected Papers 13*, pages 1–12. Springer, 2017.

[9] Max Bannach and Sebastian Berndt. Recent advances in positive-instance driven graph searching. *Algorithms*, 15(2):42, 2022.

[10] Lali Barrière, Paola Flocchini, Fedor V Fomin, Pierre Fraigniaud, Nicolas Nisse, Nicola Santoro, and Dimitrios M Thilikos. Connected graph searching. *Information and Computation*, 219:1–16, 2012.

[11] Lali Barriere, Paola Flocchini, Pierre Fraigniaud, and Nicola Santoro. Capture of an intruder by mobile agents. In *Proceedings of the fourteenth annual ACM symposium on Parallel algorithms and architectures*, pages 200–209, 2002.

[12] Lali Barriere, Pierre Fraigniaud, Nicola Santoro, and D Thilikos. Connected and internal graph searching. In *29th Workshop on Graph Theoretic Concepts in Computer Science (WG), Springer-Verlag, LNCS*, volume 2880, pages 34–45. Citeseer, 2003.

[13] Lali Barriere, Pierre Fraigniaud, Nicola Santoro, and Dimitrios M Thilikos. Searching is not jumping. In *International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 34–45. Springer, 2003.

[14] A. Bärtschi, J. Chalopin, S. Das, Y. Disser, D. Graf, J. Hackfeld, and P. Penna. Energy-efficient delivery by heterogeneous mobile agents. In *34th Symposium on Theoretical Aspects of Computer Science, STACS 2017, March 8-11, 2017, Hannover, Germany*, pages 10:1–10:14, 2017.

[15] Andreas Bärtschi. *Efficient delivery with mobile agents*. PhD thesis, ETH Zurich, 2017.

[16] Andreas Bärtschi, Evangelos Bampas, Jérémie Chalopin, Shantanu Das, Christina Karousatou, and Matúš Mihalák. Near-gathering of energy-constrained mobile agents. *Theoretical Computer Science*, 849:35–46, 2021.

[17] Andreas Bärtschi, Jérémie Chalopin, Shantanu Das, Yann Disser, Barbara Geissmann, Daniel Graf, Arnaud Labourel, and Matúš Mihalák. Collaborative delivery with energy-constrained mobile robots. *Theoretical Computer Science*, 810:2–14, 2020.

[18] Andreas Bärtschi, Daniel Graf, and Paolo Penna. Truthful mechanisms for delivery with mobile agents. *arXiv preprint arXiv:1702.07665*, 2017.

[19] Jana Bazynskiego. Searching by heterogeneous agents. In *Algorithms and Complexity: 11th International Conference, CIAC 2019, Rome, Italy, May 27–29, 2019, Proceedings*, volume 11485, page 199. Springer, 2019.

[20] Micah J Best, Arvind Gupta, Dimitrios M Thilikos, and Dimitris Zoros. Contraction obstructions for connected graph searching. *Discrete Applied Mathematics*, 209:27–47, 2016.

[21] Sayan Bhattacharya, Goutam Paul, and Swagato Sanyal. A cops and robber game in multidimensional grids. *Discrete Applied Mathematics*, 158(16):1745–1751, 2010.

[22] Dan Bienstock, Neil Robertson, Paul Seymour, and Robin Thomas. Quickly excluding a forest. *Journal of Combinatorial Theory, Series B*, 52(2):274–283, 1991.

[23] Daniel Bienstock and Michael A Langston. Algorithmic implications of the graph minor theorem. *Handbooks in Operations Research and Management Science*, 7:481–502, 1995.

[24] Daniel Bienstock and Paul Seymour. Monotonicity in graph searching. *Journal of Algorithms*, 12(2):239–245, 1991.

[25] Alain Billionnet. On interval graphs and matrice profiles. *RAIRO-Operations Research*, 20(3):245–256, 1986.

[26] Davide Bilò, Luciano Gualà, Stefano Leucci, Guido Proietti, and Mirko Rossi. New approximation algorithms for the heterogeneous weighted delivery problem. *Theoretical Computer Science*, 932:102–115, 2022.

[27] Jean Blair, Fredrik Manne, and Rodica Mihai. Efficient self-stabilizing graph searching in tree networks. In *Stabilization, Safety, and Security of Distributed Systems: 12th International Symposium, SSS 2010, New York, NY, USA, September 20-22, 2010. Proceedings 12*, pages 111–125. Springer, 2010.

[28] Lélia Blin, Janna Burman, and Nicolas Nisse. *Perpetual graph searching.* PhD thesis, INRIA, 2012.

[29] Lélia Blin, Janna Burman, and Nicolas Nisse. Exclusive graph searching. *Algorithmica*, 77(3):942–969, 2017.

[30] Lélia Blin, Pierre Fraigniaud, Nicolas Nisse, and Sandrine Vial. Distributed chasing of network intruders. *Theoretical Computer Science*, 399(1-2):12–37, 2008.

[31] H Bodlaender, L van der Gaag, and T Kloks. Some remarks on minimum edge and minimum clique triangulations. *Unpublished result.*

[32] Hans L Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth. In *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, pages 226–234, 1993.

[33] Hans L Bodlaender. A tourist guide through treewidth. *Acta cybernetica*, 11(1-2):1, 1994.

[34] Hans L Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theoretical computer science*, 209(1-2):1–45, 1998.

[35] Hans L Bodlaender. Fixed-parameter tractability of treewidth and pathwidth. In *The Multivariate Algorithmic Revolution and Beyond*, pages 196–227. Springer, 2012.

[36] Hans L Bodlaender and Ton Kloks. Efficient and constructive algorithms for the pathwidth and treewidth of graphs. *Journal of Algorithms*, 21(2):358–402, 1996.

[37] Hans L Bodlaender and Jesper Nederlof. Subexponential time algorithms for finding small tree and path decompositions. In *Algorithms-ESA 2015*, pages 179–190. Springer, 2015.

[38] Hans L Bodlaender and Dimitrios M Thilikos. Computing small search numbers in linear time. In *International Workshop on Parameterized and Exact Computation*, pages 37–48. Springer, 2004.

[39] Anthony Bonato. *The game of cops and robbers on graphs*. American Mathematical Soc., 2011.

[40] Anthony Bonato and Boting Yang. Graph searching and related problems. *Handbook of Combinatorial Optimization*, pages 1511–1558, 2013.

[41] Rolf Borchert and Norman A Slade. Bifurcation ratios and the adaptive geometry of trees. *Botanical gazette*, 142(3):394–401, 1981.

[42] Piotr Borowiecki, Shantanu Das, Dariusz Dereniowski, and Łukasz Kuszner. Distributed evacuation in graphs with multiple exits. In *International Colloquium on Structural Information and Communication Complexity*, pages 228–241. Springer, 2016.

[43] Piotr Borowiecki, Dariusz Dereniowski, and Łukasz Kuszner. Distributed graph searching with a sense of direction. *Distributed Computing*, 28:155–170, 2015.

[44] Franz J Brandenburg and Stephanie Herrmann. Graph searching and search time. In *International Conference on Current Trends in Theory and Practice of Computer Science*, pages 197–206. Springer, 2006.

[45] Richard Breisch. An intuitive approach to speleotopology. *Southwestern cavers*, 6(5):72–78, 1967.

[46] Jie Cai. *Network Decontamination from Black Viruses*. PhD thesis, Carleton University, 2016.

[47] Jie Cai, Paola Flocchini, and Nicola Santoro. Network decontamination from a black virus. In *2013 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum*, pages 696–705. IEEE, 2013.

[48] Jie Cai, Paola Flocchini, and Nicola Santoro. Black virus decontamination in arbitrary networks. In *New Contributions in Information Systems and Technologies: Volume 1*, pages 991–1000. Springer, 2015.

[49] Jérémie Chalopin, Shantanu Das, Yann Disser, Arnaud Labourel, and Matúš Mihalák. Collaborative delivery on a fixed path with homogeneous energy-constrained agents. *Theoretical Computer Science*, 868:87–96, 2021.

[50] Jérémie Chalopin, Shantanu Das, Matúš Mihal'ák, Paolo Penna, and Peter Widmayer. Data delivery by energy-constrained mobile agents. In *Algorithms for Sensor Systems: 9th International Symposium on Algorithms and Experiments for Sensor Systems, Wireless Networks and Distributed Robotics, ALGOSENSORS 2013, Sophia Antipolis, France, September 5-6, 2013, Revised Selected Papers 9*, pages 111–122. Springer, 2014.

[51] Jérémie Chalopin, Riko Jacob, Matúš Mihalák, and Peter Widmayer. Data delivery by energy-constrained mobile agents on a line. In *Automata, Languages, and Programming: 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part II 41*, pages 423–434. Springer, 2014.

[52] Jo-Mei Chang and Nicholas F. Maxemchuk. Reliable broadcast protocols. *ACM Transactions on Computer Systems (TOCS)*, 2(3):251–273, 1984.

[53] Ruay Shiung Chang. Single step graph search problem. *Information processing letters*, 40(2):107–111, 1991.

[54] Marek Chrobak, Leszek Gasieniec, and Wojciech Rytter. Fast broadcasting and gossiping in radio networks. *Journal of Algorithms*, 43(2):177–189, 2002.

[55] Timothy H Chung, Geoffrey A Hollinger, and Volkan Isler. Search and pursuit-evasion in mobile robotics: A survey. *Autonomous robots*, 31:299–316, 2011.

[56] Serafino Cicerone, Gabriele Di Stefano, Leszek Gąsieniec, and Alfredo Navarra. Asynchronous rendezvous with different maps. In *Structural Information and Communication Complexity: 26th International Colloquium, SIROCCO 2019, L'Aquila, Italy, July 1–4, 2019, Proceedings*, pages 154–169. Springer, 2019.

[57] N.E. Clarke and E.L. Connon. Cops, robber, and alarms. *Ars Comb.*, 81:283–296, 2006.

[58] N.E. Clarke and R.J. Nowakowski. Cops, robber, and photo radar. *Ars Comb.*, 56:97–103, 2000.

[59] N.E. Clarke and R.J. Nowakowski. Cops, robber and traps. *Utilitas Mathematica*, 60:91–98, 2001.

[60] Jared Coleman, Evangelos Kranakis, Danny Krizanc, and Oscar Morales-Ponce. The pony express communication problem. In *Combinatorial Algorithms: 32nd International Workshop, IWOCA 2021, Ottawa, ON, Canada, July 5–7, 2021, Proceedings 32*, pages 208–222. Springer, 2021.

[61] Stephen Cook and Ravi Sethi. Storage requirements for deterministic/polynomial time recognizable languages. In *Proceedings of the sixth annual ACM symposium on Theory of computing*, pages 33–39, 1974.

[62] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2022.

[63] Thomas Cover. Broadcast channels. *IEEE Transactions on Information Theory*, 18(1):2–14, 1972.

[64] Jerzy Czyzowicz, Krzysztof Diks, Jean Moussi, and Wojciech Rytter. Algorithms for communication problems for mobile agents exchanging energy. *arXiv preprint arXiv:1511.05987*, 2015.

[65] Jerzy Czyzowicz, Krzysztof Diks, Jean Moussi, and Wojciech Rytter. Energy-optimal broadcast in a tree with mobile agents. In *Algorithms for Sensor Systems: 13th International Symposium on Algorithms and Experiments for Wireless Sensor Networks, ALGOSENSORS 2017, Vienna, Austria, September 7-8, 2017, Revised Selected Papers 13*, pages 98–113. Springer, 2017.

[66] Jurek Czyzowicz, Dariusz Dereniowski, Robert Ostrowski, and Wojciech Rytter. Gossiping by energy-constrained mobile agents in tree networks. *Theoretical Computer Science*, 861:45–65, 2021.

[67] Jurek Czyzowicz, Krzysztof Diks, Jean Moussi, and Wojciech Rytter. Communication problems for mobile agents exchanging energy. In *Structural Information and Communication Complexity: 23rd International Colloquium, SIROCCO 2016, Helsinki, Finland, July 19-21, 2016, Revised Selected Papers 23*, pages 275–288. Springer, 2016.

[68] Jurek Czyzowicz, Krzysztof Diks, Jean Moussi, and Wojciech Rytter. Broadcast with energy-exchanging mobile agents distributed on a tree. In *Structural Information and Communication Complexity: 25th International Colloquium, SIROCCO 2018, Ma'ale HaHamisha, Israel, June 18-21, 2018, Revised Selected Papers 25*, pages 209–225. Springer, 2018.

[69] Jurek Czyzowicz, Leszek Gąsieniec, Konstantinos Georgiou, Evangelos Kranakis, and Fraser MacQuarrie. The beachcombers' problem: Walking and searching with mobile robots. *Theoretical Computer Science*, 608:201–218, 2015.

[70] Jurek Czyzowicz, Leszek Gąsieniec, Adrian Kosowski, and Evangelos Kranakis. Boundary patrolling by mobile agents with distinct maximal speeds. In *Algorithms–ESA 2011: 19th Annual European Symposium, Saarbrücken, Germany, September 5-9, 2011. Proceedings 19*, pages 701–712. Springer, 2011.

[71] Jurek Czyzowicz, Evangelos Kranakis, Dominik Pajak, and Najmeh Taleb. *Patrolling by Robots Equipped with Visibility*, pages 224–234. Springer International Publishing, Cham, 2014.

[72] Yassine Daadaa, Paola Flocchini, and Nejib Zaguia. Network decontamination with temporal immunity by cellular automata. In *Cellular Automata: 9th International Conference on Cellular Automata for Research and Industry, ACRI 2010, Ascoli Piceno, Italy, September 21-24, 2010. Proceedings 9*, pages 287–299. Springer, 2010.

[73] Yassine Daadaa, Paola Flocchini, and Nejib Zaguia. Decontamination with temporal immunity by mobile cellular automata. In *International Conference on Scientific Computing (CSC)*, pages 172–178, 2011.

[74] Yassine Daadaa, Asif Jamshed, and Mudassir Shabbir. Network decontamination with a single agent. *Graphs and Combinatorics*, 32:559–581, 2016.

[75] Shantanu Das. Graph explorations with mobile agents. *Distributed Computing by Mobile Entities: Current Research in Moving and Computing*, pages 403–422, 2019.

[76] Shantanu Das, Dariusz Dereniowski, and Christina Karousatou. Collaborative exploration by energy-constrained mobile robots. In *Structural Information and Communication Complexity: 22nd International Colloquium, SIROCCO 2015, Montserrat, Spain, July 14-16, 2015. Post-Proceedings 22*, pages 357–369. Springer, 2015.

[77] Shantanu Das, Dariusz Dereniowski, and Przemysław Uznański. Energy constrained depth first search. *arXiv preprint arXiv:1709.10146*, 2017.

[78] Anwitaman Datta, Silvia Quarteroni, and Karl Aberer. Autonomous gossiping: A self-organizing epidemic algorithm for selective information dissemination in wireless mobile ad-hoc networks. In *International Conference on Semantics for the Networked World*, pages 126–143. Springer, 2004.

[79] Yves Colin De Verdiere. Multiplicities of eigenvalues and tree-width of graphs. *Journal of Combinatorial Theory, Series B*, 74(2):121–146, 1998.

[80] Nick D Dendris, Lefteris M Kirousis, and Dimitrios M Thilikos. Fugitive-search games on graphs and related parameters. *Theoretical Computer Science*, 172(1-2):233–254, 1997.

[81] Narsingh Deo, Mukkai S Krishnamoorthy, and Michael A Langston. Exact and approximate solutions for the gate matrix layout problem. *IEEE transactions on computer-aided design of integrated circuits and systems*, 6(1):79–84, 1987.

[82] Dariusz Dereniowski. Maximum vertex occupation time and inert fugitive: Recontamination does help. *Information processing letters*, 109(9):422–426, 2009.

[83] Dariusz Dereniowski. Connected searching of weighted trees. *Theoretical Computer Science*, 412(41):5700–5713, 2011.

[84] Dariusz Dereniowski. Approximate search strategies for weighted trees. *Theoretical Computer Science*, 463:96–113, 2012.

[85] Dariusz Dereniowski. From pathwidth to connected pathwidth. *SIAM Journal on Discrete Mathematics*, 26(4):1709–1732, 2012.

[86] Dariusz Dereniowski, Öznur Yaşar Diner, and Danny Dyer. Three-fast-searchable graphs. *Discrete Applied Mathematics*, 161(13-14):1950–1958, 2013.

[87] Dariusz Dereniowski and Danny Dyer. On minimum cost edge searching. *Theoretical Computer Science*, 495:37–49, 2013.

[88] Dariusz Dereniowski, Ralf Klasing, Adrian Kosowski, and Łukasz Kuszner. Rendezvous of heterogeneous mobile agents in edge-weighted networks. *Theoretical Computer Science*, 608:219–230, 2015.

[89] Dariusz Dereniowski, Wieslaw Kubiak, and Yori Zwols. The complexity of minimum-length path decompositions. *Journal of Computer and System Sciences*, 81(8):1715–1747, 2015.

[90] Dariusz Dereniowski, Łukasz Kuszner, and Robert Ostrowski. Searching by heterogeneous agents. *Journal of Computer and System Sciences*, 115:1–21, 2021.

[91] Dariusz Dereniowski, Dorota Osula, and Paweł Rzążewski. Finding small-width connected path decompositions in polynomial time. *Theoretical Computer Science*, 794:85–100, 2019.

[92] Dariusz Dereniowski and Adam Stański. On tradeoffs between width-and fill-like graph parameters. *Theory of Computing Systems*, 63:450–465, 2019.

[93] Dariusz Dereniowski and Dorota Urbańska. On-line search in two-dimensional environment. In *International Workshop on Approximation and Online Algorithms*, pages 223–237. Springer, 2017.

[94] Anders Dessmark, Pierre Fraigniaud, Dariusz R Kowalski, and Andrzej Pelc. Deterministic rendezvous in graphs. *Algorithmica*, 46:69–96, 2006.

[95] Öznur Yaşar Diner, Danny Dyer, and Boting Yang. Four-searchable biconnected outerplanar graphs. *Discrete Applied Mathematics*, 306:70–82, 2022.

[96] Vida Dujmović, Pat Morin, and David R Wood. Path-width and three-dimensional straight-line grid drawings of graphs. In *International Symposium on Graph Drawing*, pages 42–53. Springer, 2002.

[97] Christian A Duncan, Stephen G Kobourov, and VS Anil Kumar. Optimal constrained graph exploration. *ACM Transactions on Algorithms (TALG)*, 2(3):380–402, 2006.

[98] Danny Dyer, Boting Yang, and Öznur Yaşar. On the fast searching problem. In *International Conference on Algorithmic Applications in Management*, pages 143–154. Springer, 2008.

[99] Miroslaw Dynia, Miroslaw Korzeniowski, and Christian Schindelhauer. Power-aware collective tree exploration. In *Architecture of Computing Systems - ARCS 2006, 19th International Conference, Frankfurt/Main, Germany, March 13-16, 2006, Proceedings*, pages 341–351, 2006.

[100] Gianlorenzo d'Angelo, Gabriele Di Stefano, Alfredo Navarra, Nicolas Nisse, and Karol Suchan. Computing on rings by oblivious robots: a unified approach for different tasks. *Algorithmica*, 72:1055–1096, 2015.

[101] Gianlorenzo D'angelo, Alfredo Navarra, and Nicolas Nisse. A unified approach for gathering and exclusive searching on rings under weak assumptions. *Distributed Computing*, 30:17–48, 2017.

[102] John Ellis and Robert Warren. Lower bounds on the pathwidth of some grid-like graphs. *Discrete Applied Mathematics*, 156(5):545–555, 2008.

[103] John Arthur Ellis, I Hal Sudborough, and Jonathan S Turner. *Graph separation and search number*. University of Victoria. Department of Computer Science, 1987.

[104] A. Farrugia, L. Gasieniec, Ł. Kuszner, and E. Pacheco. Deterministic rendezvous with different maps. In *Journal of Computer and System Sciences*, volume 106, pages 49–59, 2019.

[105] Ashley Farrugia, Leszek Gąsieniec, Łukasz Kuszner, and Eduardo Pacheco. Deterministic rendezvous in restricted graphs. In *SOFSEM 2015: Theory and Practice of Computer Science: 41st International Conference on Current Trends in Theory and Practice of Computer Science, Pec pod Sněžkou, Czech Republic, January 24-29, 2015. Proceedings 41*, pages 189–200. Springer, 2015.

[106] O. Feinerman, A. Korman, S. Kutten, and Y. Rodeh. Fast rendezvous on a cycle by agents with different speeds. In *Distributed Computing and Networking - 15th International Conference, ICDCN 2014, Coimbatore, India, January 4-7, 2014. Proceedings*, pages 1–13, 2014.

[107] Michael R Fellows and Michael A Langston. Nonconstructive tools for proving polynomial-time decidability. *Journal of the ACM (JACM)*, 35(3):727–739, 1988.

[108] Michael R Fellows and Michael A Langston. On search decision and the efficiency of polynomial-time algorithms. In *Proceedings of the twenty-first annual ACM symposium on Theory of computing*, pages 501–512, 1989.

[109] Michael R Fellows and Michael A Langston. On search, decision, and the efficiency of polynomial-time algorithms. *Journal of Computer and System Sciences*, 49(3):769–779, 1994.

[110] Philippe Flajolet, Jean-Claude Raoult, and Jean Vuillemin. The number of registers required for evaluating arithmetic expressions. *Theoretical Computer Science*, 9(1):99–125, 1979.

[111] Paola Flocchini, Miao Jim Huang, and Flaminia L Luccio. Decontamination of chordal rings and tori. In *Proceedings 20th IEEE International Parallel & Distributed Processing Symposium*, pages 8–pp. IEEE, 2006.

[112] Paola Flocchini, Miao Jun Huang, and Flaminia L Luccio. Contiguous search in the hypercube for capturing an intruder. In *19th IEEE International Parallel and Distributed Processing Symposium*, pages 10–pp. IEEE, 2005.

[113] Paola Flocchini, Miao Jun Huang, and Flaminia L Luccio. Decontaminating chordal rings and tori using mobile agents. *International Journal of Foundations of Computer Science*, 18(03):547–563, 2007.

[114] Paola Flocchini, Miao Jun Huang, and Flaminia L Luccio. Decontamination of hypercubes by mobile agents. *Networks: An International Journal*, 52(3):167–178, 2008.

[115] Paola Flocchini, Fabrizio Luccio, Linda Pagli, and Nicola Santoro. Optimal network decontamination with threshold immunity. In *Algorithms and Complexity: 8th International Conference, CIAC 2013, Barcelona, Spain, May 22-24, 2013. Proceedings 8*, pages 234–245. Springer, 2013.

[116] Paola Flocchini, Fabrizio Luccio, Linda Pagli, and Nicola Santoro. Network decontamination under m-immunity. *Discrete Applied Mathematics*, 201:114–129, 2016.

[117] Paola Flocchini, Flaminia L Luccio, and Lisa Xiuli Song. Size optimal strategies for capturing an intruder in mesh networks. 2005.

[118] Paola Flocchini, Bernard Mans, and Nicola Santoro. Tree decontamination with temporary immunity. In *International Symposium on Algorithms and Computation*, pages 330–341. Springer, 2008.

[119] Paola Flocchini, Amiya Nayak, and Arno Schulz. Cleaning an arbitrary regular network with mobile agents. In *International Conference on Distributed Computing and Internet Technology*, pages 132–142. Springer, 2005.

[120] Paola Flocchini, Amiya Nayak, and Arno Schulz. Decontamination of arbitrary networks using a team of mobile agents with limited visibility. In *6th IEEE/ACIS International Conference on Computer and Information Science (ICIS 2007)*, pages 469–474. IEEE, 2007.

[121] Paola Flocchini and Nicola Santoro. Distributed security algorithms for mobile agents. *Mobile Agents in Networking and Distributed Computing*, pages 41–70, 2012.

[122] Paola Flocchini, Nicola Santoro, and Lisa Xiuli Song. On the complexity of decontaminating an hexagonal mesh network. In *2007 International Multi-Conference on Computing in the Global Information Technology (ICCGI'07)*, pages 21–21. IEEE, 2007.

[123] Paola Flocchini, Nicola Santoro, Giovanni Viglietta, and Masafumi Yamashita. Rendezvous with constant memory. *Theoretical Computer Science*, 621:57–72, 2016.

[124] Fedor V Fomin. Complexity of connected search when the number of searchers is small. *Open problems of GRASTA*, 2017.

[125] Fedor V Fomin, Pierre Fraigniaud, and Nicolas Nisse. Nondeterministic graph searching: From pathwidth to treewidth. *Algorithmica*, 53(3):358–373, 2009.

[126] Fedor V Fomin and Petr A Golovach. Graph searching and interval completion. *SIAM Journal on Discrete Mathematics*, 13(4):454–464, 2000.

[127] Fedor V Fomin and Dimitrios M Thilikos. On the monotonicity of games generated by symmetric submodular functions. *Discrete Applied Mathematics*, 131(2):323–335, 2003.

[128] Fedor V Fomin and Dimitrios M Thilikos. An annotated bibliography on guaranteed graph searching. *Theoretical computer science*, 399(3):236–245, 2008.

[129] Pierre Fraigniaud, David Ilcinkas, and Andrzej Pelc. Oracle size: a new measure of difficulty for communication tasks. In *Proceedings of the twenty-fifth annual ACM symposium on Principles of distributed computing*, pages 179–187, 2006.

[130] Pierre Fraigniaud and Nicolas Nisse. Connected treewidth and connected graph searching. In *LATIN 2006: Theoretical Informatics: 7th Latin American Symposium, Valdivia, Chile, March 20-24, 2006. Proceedings 7*, pages 479–490. Springer, 2006.

[131] Pierre Fraigniaud and Nicolas Nisse. Monotony properties of connected visible graph searching. *Information and Computation*, 206(12):1383–1393, 2008.

[132] Pierre Fraigniaud and Sandrine Vial. Approximation algorithms for broadcasting and gossiping. *Journal of Parallel and Distributed Computing*, 43(1):47–55, 1997.

[133] Matthew Franklin, Zvi Galil, and Moti Yung. Eavesdropping games: a graph-theoretic approach to privacy in distributed systems. *Journal of the ACM (JACM)*, 47(2):225–243, 2000.

[134] Leszek Gąsieniec and Igor Potapov. Gossiping with unit messages in known radio networks. In *Foundations of Information Technology in the Era of Network and Mobile Computing: IFIP 17th World Computer Congress—TC1 Stream/2nd IFIP International Conference on Theoretical Computer Science (TCS 2002) August 25–30, 2002, Montréal, Québec, Canada*, pages 193–205. Springer, 2002.

[135] Leszek Gąsieniec, Igor Potapov, and Qin Xin. Time efficient gossiping in known radio networks. In *International Colloquium on Structural Information and Communication Complexity*, pages 173–184. Springer, 2004.

[136] Aristotelis Giannakos, Mhand Hifi, and Gregory Karagiorgos. Data delivery by mobile agents with energy constraints over a fixed path. *arXiv preprint arXiv:1703.05496*, 2017.

[137] Archontia C Giannopoulou, Paul Hunter, and Dimitrios M Thilikos. Lifo-search: A min–max theorem and a searching game for cycle-rank and tree-depth. *Discrete Applied Mathematics*, 160(15):2089–2097, 2012.

[138] John R Gilbert, Thomas Lengauer, and Robert Endre Tarjan. The pebbling problem is complete in polynomial space. In *Proceedings of the eleventh annual ACM symposium on Theory of computing*, pages 237–248, 1979.

[139] Alexander Gleyzer, Michael Denisyuk, Alon Rimmer, and Yigal Salingar. A fast recursive gis algorithm for computing strahler stream order in braided and nonbraided networks 1. *JAWRA Journal of the American Water Resources Association*, 40(4):937–946, 2004.

[140] Petr A Golovach. Equivalence of 2 formalizations of the search problem in a graph. *Vestnik Leningradskogo Universiteta Seriya Matematika Mekhanika Astronomiya*, (1):10–14, 1989. Translation in translation in Vestnik Leningrad Univ. Math. 22 (1989), no. 1, 13–19.

[141] Petr A Golovach. A topological invariant in pursuit problems. *Differentsial'nye Uravneniya*, 25(6):923–929, 1989. Translation in Differ. Equ. 25 (1989), no. 6, 657–661.

[142] Georg Gottlob, Zoltán Miklós, and Thomas Schwentick. Generalized hypertree decompositions: Np-hardness and tractable variants. *Journal of the ACM (JACM)*, 56(6):1–32, 2009.

[143] András Hajnal, Eric C Milner, and Endre Szemerédi. A cure for the telephone disease. *Canadian Mathematical Bulletin*, 15(3):447–450, 1972.

[144] Daniel John Harvey. *On treewidth and graph minors*. PhD thesis, University of Melbourne, Department of Mathematics and Statistics, 2014.

[145] Sandra Mitchell Hedetniemi, Stephen T. Hedetniemi, and Arthur L. Liestman. A survey of gossiping and broadcasting in communication networks. *Networks*, 18(4):319–349, 1988.

[146] Philipp Hertel and Toniann Pitassi. The pspace-completeness of black-white pebbling. *SIAM Journal on Computing*, 39(6):2622–2682, 2010.

[147] G.A. Hollinger, A. Kehagias, and S. Singh. GSST: anytime guaranteed search. *Auton. Robots*, 29(1):99–118, 2010.

[148] Geoffrey Hollinger, Sanjiv Singh, Joseph Djugash, and Athanasios Kehagias. Efficient multi-robot search for a moving target. *The International Journal of Robotics Research*, 28(2):201–219, 2009.

[149] Keith Horsfield. Some mathematical properties of branching trees with application to the respiratory system. *Bulletin of mathematical biology*, 38(3):305–315, 1976.

[150] Robert E Horton. Erosional development of streams and their drainage basins; hydrophysical approach to quantitative morphology. *Geological society of America bulletin*, 56(3):275–370, 1945.

[151] Juraj Hromkovič, Ralf Klasing, Andrzej Pelc, Peter Ruzicka, and Walter Unger. *Dissemination of information in communication networks: broadcasting, gossiping, leader election, and fault-tolerance*. Springer Science & Business Media, 2005.

[152] Ju Yuan Hsiao, Chuan Yi Tang, and Ruay Shiung Chang. Solving the single step graph searching problem by solving the maximum two-independent set problem. *Information processing letters*, 40(5):283–287, 1991.

[153] Ju Yuan Hsiao, Chuan Yi Tang, and Ruay Shiung Chang. The summation and bottleneck minimization for single-step searching on weighted graphs. *Information sciences*, 74(1-2):1–28, 1993.

[154] Ju Yuan Hsiao, Chuan Yi Tang, Ruay Shiung Chang, and Richard C. T. Lee. Single step searching in weighted block graphs. *Information sciences*, 81(1-2):1–29, 1994.

[155] Miao Jun Huang. *Contiguous search by mobile agents in cube networks and chordal rings*. PhD thesis, University of Ottawa (Canada), 2004.

[156] Glenn Hurlbert. Graph pebbling. *Handbook of Graph Theory, Ed: JL Gross, J. Yellen, P. Zhang, Chapman and Hall/CRC, Kalamazoo*, pages 1428–1449, 2013.

[157] David Ilcinkas, Nicolas Nisse, and David Soguet. The cost of monotonicity in distributed graph searching. *Distributed Computing*, 22:117–127, 2009.

[158] Navid Imani, Hamid Sarbazi-Azad, and Albert Y Zomaya. Capturing an intruder in product networks. *Journal of Parallel and Distributed Computing*, 67(9):1018–1028, 2007.

[159] Navid Imani, Hamid Sarbazi-Azad, Albert Y Zomaya, and Parya Moinzadeh. Detecting threats in star graphs. *IEEE Transactions on Parallel and Distributed Systems*, 20(4):474–483, 2008.

[160] Mamadou M Kanté, Christophe Paul, and Dimitrios M Thilikos. A linear fixed parameter tractable algorithm for connected pathwidth. *SIAM Journal on Discrete Mathematics*, 36(1):411–435, 2022.

[161] Akitoshi Kawamura and Yusuke Kobayashi. Fence patrolling by mobile agents with distinct speeds. *Distributed Computing*, 28:147–154, 2015.

[162] Alex Kesselman and Dariusz Kowalski. Fast distributed algorithm for convergecast in ad hoc geometric radio networks. In *Second Annual Conference on Wireless On-demand Network Systems and Services*, pages 119–124. IEEE, 2005.

[163] Jonghoek Kim. Distributed rendezvous of heterogeneous robots with motion-based power level estimation. *Journal of Intelligent & Robotic Systems*, 100(3-4):1417–1427, 2020.

[164] Nancy G Kinnersley. The vertex separation number of a graph equals its path-width. *Information Processing Letters*, 42(6):345–350, 1992.

[165] Nancy Gail Kinnersley. *Obstruction set isolation for layout permutation problems*. Washington State University, 1989.

[166] Lefteris M Kirousis and Christos H Papadimitriou. Interval graphs and searching. *Discrete Mathematics*, 55(2):181–184, 1985.

[167] Lefteris M Kirousis and Christos H Papadimitriou. Searching and pebbling. *Theoretical Computer Science*, 47:205–218, 1986.

[168] Lefteris M Kirousis and Dimitris M Thilikos. The linkage of a graph. *SIAM Journal on Computing*, 25(3):626–647, 1996.

[169] Ton Kloks. *Treewidth: computations and approximations*. Springer, 1994.

[170] András Kornai and Zsolt Tuza. Narrowness, pathwidth, and their application in natural language processing. *Discrete Applied Mathematics*, 36(1):87–92, 1992.

[171] Andrea S LaPaugh. Recontamination does not help to search a graph. *Journal of the ACM (JACM)*, 40(2):224–245, 1993.

[172] Hoàng-Oanh Le, Haiko Müller, et al. Splitting a graph into disjoint induced paths or cycles. *Discrete applied mathematics*, 131(1):199–212, 2003.

[173] Bi Li, Fatima Zahra Moataz, Nicolas Nisse, and Karol Suchan. Minimum size tree-decompositions. *Electronic Notes in Discrete Mathematics*, 50:21–27, 2015.

[174] Yichao Lin. *Decontamination from black viruses using parallel strategies*. PhD thesis, Université d'Ottawa/University of Ottawa, 2018.

[175] Andrzej Lingas. A pspace complete problem related to a pebble game. In *International Colloquium on Automata, Languages, and Programming*, pages 300–321. Springer, 1978.

[176] Fabrizio Luccio and Linda Pagli. A general approach to toroidal mesh decontamination with local immunity. In *2009 IEEE International Symposium on Parallel & Distributed Processing*, pages 1–8. IEEE, 2009.

[177] Fabrizio Luccio, Linda Pagli, and Nicola Santoro. Network decontamination in presence of local immunity. *International Journal of Foundations of Computer Science*, 18(03):457–474, 2007.

[178] Flaminia L Luccio. Contiguous search problem in sierpiński graphs. *Theory of Computing Systems*, 44:186–204, 2009.

[179] G.A. Di Luna, P. Flocchini, N. Santoro, G. Viglietta, and M. Yamashita. Self-stabilizing meeting in a polygon by anonymous oblivious robots. *CoRR*, abs/1705.00324, 2017.

[180] Eva Ma and Lixin Tao. Embeddings among toruses and meshes. *Technical Reports (CIS)*, page 598, 1988.

[181] Fillia S Makedon, Christos H Papadimitriou, and Ivan Hal Sudborough. Topological bandwidth. *SIAM Journal on Algebraic Discrete Methods*, 6(3):418–444, 1985.

[182] FS Makedon and Ivan Hal Sudborough. Minimizing width in linear layouts. In *International Colloquium on Automata, Languages, and Programming*, pages 478–490. Springer, 1983.

[183] Euripides Markou, Nicolas Nisse, and Stéphane Pérennes. Exclusive graph searching vs. pathwidth. *Information and Computation*, 252:243–260, 2017.

[184] Frédéric Mazoit and Nicolas Nisse. Monotonicity of non-deterministic graph searching. *Theoretical Computer Science*, 399(3):169–178, 2008.

[185] Stephen D McCormick, Anthony P Farrell, and Colin J Brauner. *Fish physiology: euryhaline fishes*. Academic Press, 2013.

[186] Nimrod Megiddo, S Louis Hakimi, Michael R Garey, David S Johnson, and Christos H Papadimitriou. The complexity of searching a graph. *Journal of the ACM (JACM)*, 35(1):18–44, 1988.

[187] Guillaume Mescoff, Christophe Paul, and Dimitrios M Thilikos. The mixed search game against an agile and visible fugitive is monotone. *Discrete Mathematics*, 346(4):113345, 2023.

[188] Margaret-Ellen Messinger. *Methods of decontaminating networks*. PhD thesis, Dalhousie University, 2008.

[189] Rodica Mihai and Morten Mjelde. A self-stabilizing algorithm for graph searching in trees. In *Stabilization, Safety, and Security of Distributed Systems: 11th International Symposium, SSS 2009, Lyon, France, November 3-6, 2009. Proceedings 11*, pages 563–577. Springer, 2009.

[190] Rodica Mihai and Ioan Todinca. Pathwidth is NP-hard for weighted trees. In *FAW '09: Proc. of the 3rd Inter. Workshop on Frontiers in Algorithmics*, pages 181–195, Berlin, Heidelberg, 2009. Springer-Verlag.

[191] Jaroslav Nešetřil and Patrice Ossona De Mendez. Tree-depth, subgraph coloring and homomorphism bounds. *European Journal of Combinatorics*, 27(6):1022–1041, 2006.

[192] Lex Newman. Descartes' epistemology: Evil genius doubt, 2023. `https://plato.stanford.edu/entries/descartes-epistemology/#EvilGeniDoub` [Accessed: (14.02.2024)].

[193] Nicolas Nisse. Connected graph searching in chordal graphs. *Discrete Applied Mathematics*, 157(12):2603–2610, 2009.

[194] Nicolas Nisse. Network decontamination. In *Distributed Computing by Mobile Entities: Current Research in Moving and Computing*, pages 516–548. Springer, 2019.

[195] Nicolas Nisse and David Soguet. Graph searching with advice. *Theoretical Computer Science*, 410(14):1307–1318, 2009.

[196] Robert Ostrowski. Non-monotone graph searching models. *Zeszyty Naukowe Wydziału ETI Politechniki Gdańskiej. Technologie Informacyjne*, (23):90–96, 2018.

[197] Dorota Osula. *Multi-agent graph searching and exploration algorithms*. PhD thesis, Politechnika Gdańska, 2020.

[198] Christos H Papadimitriou. Computational complexity. In *Encyclopedia of computer science*, pages 260–265. 2003.

[199] Torrence D Parsons. Pursuit-evasion in a graph. In *Theory and applications of graphs*, pages 426–441. Springer, 1978.

[200] Andrzej Pelc. Deterministic rendezvous algorithms. In *Distributed Computing by Mobile Entities: Current Research in Moving and Computing*, pages 423–454. Springer, 2019.

[201] Sheng-Lung Peng, Chin-Wen Ho, Tsan-sheng Hsu, Ming-Tat Ko, and Chuan Yi Tang. Edge and node searching problems on trees. *Theoretical Computer Science*, 240(2):429–446, 2000.

[202] Nikolai Nikolaevich Petrov. A problem of pursuit in the absence of information on the pursued. *Differentsial'nye Uravneniya*, 18(8):1345–1352, 1982.

[203] Z. Qian, J. Li, X. Li, M. Zhang, and H. Wang. Modeling heterogeneous traffic flow: A pragmatic approach. *Transportation Research Part B: Methodological*, 99:183–204, 2017.

[204] Jun Qiu. *Best effort decontamination of networks*. PhD thesis, University of Ottawa (Canada), 2007.

[205] Livaniaina Hary Rakotomalala. *Network Decontamination using Cellular Automata*. PhD thesis, Université d'Ottawa/University of Ottawa, 2016.

[206] Neil Robertson and Paul D Seymour. Graph minors. i. excluding a forest. *Journal of Combinatorial Theory, Series B*, 35(1):39–61, 1983.

[207] Neil Robertson and Paul D. Seymour. Graph minors. ii. algorithmic aspects of tree-width. *Journal of algorithms*, 7(3):309–322, 1986.

[208] Neil Robertson and Paul D Seymour. Graph minors. xx. wagner's conjecture. *Journal of Combinatorial Theory, Series B*, 92(2):325–357, 2004.

[209] Daniel Preston Sanders. *Linear algorithms for graphs of tree-width at most four*. Georgia Institute of Technology, 1993.

[210] Nicola Santoro. *Design and analysis of distributed algorithms*. John Wiley & Sons, 2006.

[211] Petra Scheffler. A linear algorithm for the pathwidth of trees. In *Topics in combinatorics and graph theory*, pages 613–620. Springer, 1990.

[212] Ravi Sethi. Complete register allocation problems. In *Proceedings of the fifth annual ACM symposium on Theory of computing*, pages 182–195, 1973.

[213] Paul D Seymour and Robin Thomas. Graph searching and a min-max theorem for tree-width. *Journal of Combinatorial Theory, Series B*, 58(1):22–33, 1993.

[214] Pooya Shareghi, Navid Imani, and Hamid Sarbazi-Azad. Capturing an intruder in the pyramid. In *Computer Science–Theory and Applications: First International Computer Science Symposium in Russia, CSR 2006, St. Petersburg, Russia, June 8-12. 2006. Proceedings 1*, pages 580–590. Springer, 2006.

[215] Lisa Xiuli Song. *Intruder capture by mobile agents in mesh topologies*. PhD thesis, Carleton University, 2005.

[216] S Sreedevi and MS Anilkumar. a comprehensive review on graph pebbling and rubbling. In *Journal of Physics: Conference Series*, volume 1531, page 012050. IOP Publishing, 2020.

[217] Donald Stanley and Boting Yang. Fast searching games on graphs. *Journal of combinatorial optimization*, 22(4):763–777, 2011.

[218] Arthur N Strahler. Hypsometric (area-altitude) analysis of erosional topography. *Geological society of America bulletin*, 63(11):1117–1142, 1952.

[219] Arthur N Strahler. Quantitative analysis of watershed geomorphology. *Eos, Transactions American Geophysical Union*, 38(6):913–920, 1957.

[220] S. Sundaram, K. Krishnamoorthy, and D.W. Casbeer. Pursuit on a graph under partial information from sensors. *CoRR*, abs/1609.03664, 2016.

[221] Ichiro Suzuki and Masafumi Yamashita. Distributed anonymous mobile robots: Formation of geometric patterns. *SIAM Journal on Computing*, 28(4):1347–1363, 1999.

[222] Amnon Ta-Shma and Uri Zwick. Deterministic rendezvous, treasure hunts and strongly universal exploration sequences. In *Symposium on Discrete Algorithms: Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, volume 7, pages 599–608, 2007.

[223] Atsushi Takahashi, Shuichi Ueno, and Yoji Kajitani. Minimal acyclic forbidden minors for the family of graphs with bounded path-width. *Discrete Mathematics*, 127(1-3):293–304, 1994.

[224] Atsushi Takahashi, Shuichi Ueno, and Yoji Kajitani. Mixed searching and proper-path-width. *Theoretical Computer Science*, 137(2):253–268, 1995.

[225] Hisao Tamaki. Positive-instance driven dynamic programming for treewidth. *Journal of Combinatorial Optimization*, 37(4):1283–1311, 2019.

[226] Dimitrios M Thilikos. Algorithms and obstructions for linear-width and related search parameters. *Discrete Applied Mathematics*, 105(1-3):239–271, 2000.

[227] Paolo Toth and Daniele Vigo. *Vehicle routing: problems, methods, and applications*. SIAM, 2014.

[228] DAN WEINER. Forbidden minors and minor-closed graph properties. *Lecture notes*.

[229] Douglas Brent West et al. *Introduction to graph theory*, volume 2. Prentice hall Upper Saddle River, 2001.

[230] Chris Worman and Boting Yang. Searching trees with sources and targets. In *FAW '08: Proc. of the 2nd annual international workshop on Frontiers in Algorithmics*, pages 174–185, Berlin, Heidelberg, 2008. Springer-Verlag.

[231] Yuan Xue and Boting Yang. The fast search number of a cartesian product of graphs. *Discrete Applied Mathematics*, 224:106–119, 2017.

[232] Yuan Xue, Boting Yang, Farong Zhong, and Sandra Zilles. Fast searching on complete k-partite graphs. In *International Conference on Combinatorial Optimization and Applications*, pages 159–174. Springer, 2016.

[233] Takahiro Yakami, Yukiko Yamauchi, Shuji Kijima, and Masafumi Yamashita. Searching for an evader in an unknown dark cave by an optimal number of asynchronous searchers. *Theoretical Computer Science*, 887:11–29, 2021.

[234] Boting Yang. Strong-mixed searching and pathwidth. *Journal of combinatorial optimization*, 13(1):47–59, 2007.

[235] Boting Yang. Fast edge searching and fast searching on graphs. *Theoretical Computer Science*, 412(12-14):1208–1219, 2011.

[236] Boting Yang. Fast–mixed searching and related problems on graphs. *Theoretical Computer Science*, 507:100–113, 2013.

[237] Boting Yang, Danny Dyer, and Brian Alspach. Sweeping graphs with large clique number. *Discrete Mathematics*, 309(18):5770–5780, 2009.

[238] Daadaa Yassine. *Network decontamination with temporal immunity*. University of Ottawa (Canada), 2012.

[239] Dimitris Zoros. *Obstructions and algorithms for graph layout problems*. PhD thesis, 2017.