Collaboration in Distributed Systems: Robots, Ants, and Matchings

A thesis submitted to attain the degree of

DOCTOR OF SCIENCES of ETH ZURICH

(Dr. sc. ETH Zurich)

presented by

TOBIAS LANGNER

MSc, University of Freiburg, Germany

born on 18.11.1982

citizen of

Germany

2015

Abstract

From robots playing soccer together, through ants ensuring the survival of their colony, to nodes in a peer-to-peer network, collaboration is a corner-stone of the success of many different kind of distributed systems. The goal of this thesis is to shed more light onto various kinds of collaboration, and the advantages that individuals get from working together with others.

In the first part of the thesis, we consider a version of the Gale-Shapley stable matching setting, where each pair of n nodes is associated with a (symmetric) matching cost and the preferences are determined with respect to these costs. This stable matching version is analyzed through the Price of Anarchy (PoA) and Price of Stability (PoS) lens with the objective of minimizing the total cost of matched nodes. A simple example demonstrates that in the general case, the situation is hopeless, hence we restrict our attention to metric costs. Our first result is a tight bound of $\Theta(n^{\log(3/2)})$ on the PoA in such metric graphs. We then use the notion of α-stability, where a pair of unmatched nodes defect only if both can thereby reduce their costs by a factor greater than $\alpha \geq 1$. Our main result is an asymptotically tight trade-off, showing that with respect to α-stable matchings, the PoS is $\Theta(n^{\log(1+\frac{1}{2\alpha})})$. The proof is constructive: we present a simple algorithm that outputs an α-stable matching satisfying this bound.

In the second part of the dissertation, we examine various aspects of systems of mobile robots (or agents) with restricted capabilities. We analyze an existing gathering algorithm for n robots that cannot communicate with each other and have only limited visibility, and show that the algorithm has polynomial runtime of $\Theta(n^2)$. We then turn our view towards n agents controlled by finite automata that explore a two-dimensional integer grid while being able to communicate with each other in a very restricted manner. Their goal is to locate a treasure which is located in some cell at a distance D from the common starting point of all agents. Despite the restriction to constant-size memory, we show that their collaborative performance is sufficient to locate the treasure in an optimal time of $\mathcal{O}(D + D^2/n)$ even in an asynchronous setting. We also look at small, i.e., constant numbers of agents and give upper and lower bounds on the minimal number of ants sufficient to locate the treasure for various modifications of the aforementioned model.

In the last part of the thesis, we focus on the power of teamwork in systems that actively try to prevent collaboration. We investigate the feasibility of a Sybil attack, where many fake identities are created in order to get an unfair advantage, against online poker platforms. For this purpose, we implemented a large-scale attack on a poker platform in which automated players (bots) collaborate to increase their chances of winning. Due to ethical considerations, our bots were only deployed at play money tables, where we found that there is a linear rise in the average gain when increasing the number of bots. We conjecture that the essence of our findings can be generalized to real money tables and conclude that

it is indeed possible to benefit from such an attack and that poker platforms are in dire need of stronger countermeasures.

On the whole, the quintessence of this dissertation is that collaboration, when implemented properly, is a versatile and effective instrument to improve the performance of all kinds of distributed systems in various ways.

Zusammenfassung

Von Fussball spielenden Robotern über Ameisen, die das Überleben ihrer Kolonie sicherstellen, zu Knoten in einem Peer-To-Peer-Netzwerk: Kollaboration ist einer der Eckpfeiler verschiedenster Erfolgsgeschichten verteilter Systeme. Das Ziel dieser Arbeit ist es, verschiedene Arten von Kollaboration näher zu beleuchten sowie die Vorteile von Individuen herauszuarbeiten, die miteinander zusammen arbeiten

Im ersten Teil dieser Arbeit betrachten wir eine Variante des stabilen Matchings nach Gale-Shapley, in welcher jedem Paar aus einer Menge von n Knoten (symmetrische) Kosten zugeordnet werden, und die Präferenzen für einander bezüglich dieser Kosten bestimmt werden. Diese Version des stabilen Matchings wird mittels der Price of Anarchy- (PoA) und Price of Stability-Konzepte (PoS) analysiert, mit dem Ziel die Gesamtkosten der miteinander gepaarten Knoten zu minimieren. Ein einfaches Beispiel zeigt, dass die Situation im allgemeinen Fall hoffnungslos ist, weshalb wir uns auf metrische Kosten beschränken. Unser erstes Ergebnis ist eine scharfe Schranke von $\Theta(n^{\log(3/2)})$ für PoA und PoS in metrischen Graphen. Anschliessend betrachten wir α-Stabilität, wobei ein Paar von nicht einander zugeordneten Knoten nur dann instabil ist, wenn durch einen Wechsel der Partner beide ihre Kosten um mindestens einen Faktor $\alpha \geq 1$ reduzieren können. Unser Hauptergebnis ist ein asymptotisch scharfer Trade-Off für PoS im Bezug auf α-Stabilität von $\Theta(n^{\log(1+\frac{1}{2\alpha})})$. Wir präsentieren einen konstruktiven Beweis in der Form eines einfachen Algorithmus, der ein α-stabiles Matching bestimmt, welches diese Schranke realisiert.

Im zweiten Teil dieser Dissertation untersuchen wir verschiedene Aspekte von Systemen mobiler Roboter (oder Agenten) mit beschränkten Fähigkeiten. Wir analysieren einen bereits existierenden Algorithmus zum Sammeln von n Robotern in einem Punkt der Ebene, wobei sich die Roboter innerhalb beschränkter Sichtweite sehen, nicht aber miteinander kommunizieren können. Wir beweisen, dass die Laufzeit des Algorithmus polynomiell in n ist, indem wir eine scharfe Schranke von $\Theta(n^2)$ zeigen. Dann wenden wir uns n von endlichen Automaten gesteuerten Agenten zu, die das zwei-dimensionale ganzzahlige Gitter erkunden und nur auf sehr rudimentäre Art und Weise miteinander kommunizieren können. Ihr Ziel ist es, einen Schatz zu finden, der sich in einer Zelle befindet, die im Abstand D vom gemeinsamen Startpunkt aller Agenten liegt. Wir zeigen, dass die Agenten trotz der Beschränkung auf eine konstante Speichergrösse in der Lage sind, kollaborativ den Schatz innerhalb einer optimalen Laufzeit von $\Theta(D + D^2/n)$ zu finden und dass dies sogar in einem asynchronen Szenario gilt. Weiterhin betrachten wir kleine, d. h. konstante Anzahlen von Agenten und zeigen für verschiedene Modifikationen des beschriebenen Modells obere und untere Schranken für die minimale Anzahl an Agenten, die ausreichend ist, um den Schatz zu finden.

Im letzten Teil der Arbeit betrachten wir den Einfluss von Zusammenarbeit in Systemen, die Kollaboration aktiv zu verhindern versuchen. Wir untersuchen

die Machbarkeit eines sogenannten Sybil-Angriffs, bei welchem viele Kopien einer Identität erzeugt werden, um dadurch einen unfairen Vorteil zu erlangen, auf eine Online-Poker-Plattform. Zu diesem Zweck implementierten wir einen grossangelegten Angriff auf eine Poker-Plattform, in welchem automatisierte Spieler (Bots) am selben Tisch spielen und miteinander kollaborieren, um dadurch ihre Gewinnchancen zu erhöhen. In Anbetracht ethischer Gesichtspunkte entschieden wir uns die Bots nur an Spielgeldtischen einzusetzen, wo wir einen linearen Zusammenhang zwischen dem Durchschnittsgewinn pro Bot und der Anzahl der Bots am Tisch feststellen konnten. Wir mutmassen, dass sich die Essenz unserer Ergebnisse auch auf Echtgeldtische übertragen lässt und kommen daher zum Schluss, dass Pokerplattformen in der Tat dringend stärkere Gegenmassnahmen gegen derartige Angriffe nötig haben.

Insgesamt betrachtet besteht die Kernaussage dieser Dissertation darin, dass Kollaboration ein vielseitiges und effektives Werkzeug ist, um die Leistungsfähigkeit vielerlei Arten von verteilten Systemen auf verschiedene Weisen zu verbessern.

Acknowledgements

My time as a PhD student in the Distributed Computing Group was an instructive and exciting experience, despite also sometimes being quite stressful and hectic. In particular, I enjoyed the opportunity to explore the depths of projects that personally interested me completely on my own. However, the thesis that you are now reading would not have been possible without the help of many people that I would like to thank in the following.

First, I want to thank my supervisor Roger Wattenhofer for giving me the opportunity to write my thesis in his group and supporting me throughout the time. He guided me through the treacherous labyrinth called academia and proved to be an endless source of new promising research questions. Despite our occasional disagreements, he taught me countless important lessons, among them how to give exceptional presentations.

Then, I would also like to thank my co-referees Yuval Emek and Franck Petit for taking the time to review this thesis and to serve on my committee. Besides being one of my co-referees, Yuval was also supervisor-in-charge for one year of my studies and I have learned so many things from him, most importantly how to properly write a scientific article (from which you will hopefully benefit in the rest of the thesis).

Furthermore, there are also the other people that made my time in the Distributed Computing Group a wonderful experience; my colleagues and co-workers. I want to thank (in alphabetical order) Barbara Keller for teaching me how to cook amazing food without bacon, Beat Futterknecht for being the most efficient problem-solver on the planet, Benny Gächter for resuscitating our IT infrastructure, Christian Decker for being my last resort for Git and Python problems, Christoph Lenzen for being an amazingly funny office mate, David Stolz for inventing the super ants, Jara Uitto for importing the no-pants policy, Jasmin Smula for being a loyal Finnish ice-running mate, Jochen Seidel for hosting the Maus-Frühstück, Johannes Schneider for introducing me to Frau Muther, Klaus Förster for being the one and only Captain, Laura Peer for destroying her iPhone in an elegant manner, Léonard von Niederhäusern for being our 80s music expert (despite not even having lived a single day in the 80s), The King (Michael König) for his witty sense of humor, Michael Kuhn for being a fan of the wrong FCB, Panda Metaiel for giving me moral support during conference talks, Pascal Bissig for his unique contributions to the coffee break, Philipp Brandes for being the most responsible CLO (Chief Lunch Officer) of all times, Raphael Eidenbenz for teaching me the beauty of Bündnerdeutsch, Friederike Brütsch for paying all

my bills, Samuel Welten for inventing the FAB newsletter, Sebastian Brandt for sparking my interest in chess, Stephan Holzer for offering me a place to live in my first days in Zurich, and Yuezhou Lv for being a true expert on sarcasm.

Then there also were a few friends that helped me with proof-reading my thesis and I am very grateful to Jara Uitto, Philipp Brandes, Sebastian Brandt, and Tolga Goren for their helpful comments and suggestions.

During my PhD studies, I supervised bachelor, master, and group projects of many aspiring students and I enjoyed this diversion from the sometimes frustrating research tasks very much. I want to thank (again alphabetically) Adrian Leuenberger, Benjamin Zehnder, Christian Cadruvi, Damian Pfammatter, Dominic Plangger, Dominik Landtwing, Fabian Mentzner, Gian Ulli, Hildur Ólafsdóttir, Jan Bernegger, Jan Schulze, Kevin Jeisy, Lukas Affolter, Marc Hüppin, Marian Runo, Martin Ambühl, Nils Reinthaler, Nicolas Forster, Pascal Fischli, Pascal Niklaus, Patrick Misteli, Roni Häcki, Samuel Zihlmann, Sandro Affentranger, Sebastian Wendland, Simon Wehrli, Stephan Dollberg, Theodoros Bourchas, Timon Ruban, and Ueli Ebnöther.

Last but not least, I want to thank the people most important in my life: My parents Hartmut and Marlene for raising me as a (mostly) responsible person and for never surrendering to the endless stream of questions that my curiosity generated in my youth; my dear sister Kristiane for upholding the tradition of somewhat regular phone calls also when I was too busy to think of it and for always lending me an ear when I am in need; and my wonderful girlfriend Katarina whose unwavering love and support helped me to get through the more difficult times of my PhD studies. Without you guys, I would not be where I am now, thank you very much!

Contents

1

Introduction

"Alone we can do so little; together we can do so much."

— Helen Keller

This quote by the famous author and political activist Helen Keller who, as an aside, was the first deaf-blind person in history to earn a bachelor of arts degree, beautifully summarizes in just one sentence the essence of this work. The goal of this thesis is to shed more light onto the benefits (see Chapters 2–5) and drawbacks (see Chapter 6) of collaboration, and onto the difficulties that arise when many entities have to work together in unison.

Collaboration is indeed at the very heart of many "success"[1] stories that surround us in our life. There are the rather obvious instances of collaboration, such as the gigantic international projects established to design and construct the world's largest and most powerful particle accelerator, CERN, or the largest artificial object in an orbit around the globe, the International Space Station. In both cases, hundreds or even thousands of people from more than ten different countries tirelessly worked together for several decades to eventually make history. Then there are also more subtle appearances of collaboration; as simple as a trade between entities, be it a person buying a hot dog on the street from a vendor, or, on the other end of the spectrum, the multi-billion dollar sale of the company WhatsApp to Marc Zuckerberg's Facebook in spring 2014. Despite its apparent greedy nature, trade is in most cases a collaborative effort providing mutual benefits to both trade partners.

Moving closer towards the topic discussed in the first part of this thesis, a relationship between two or more partners clearly requires collaborative efforts

[1] Not everyone may agree that the following examples were all actually successes, hence the quotation marks.

of all involved parties in order to last. This setting is particularly interesting from a computer science perspective as the stability of a relationship — the combined incentive to continue the relationship vs. either of the partners searching for better partner — can be elegantly modeled in a graph-theoretic matching setting. The notion of stability translates to this setting in the sense that a set of relationships is stable if there are no two individuals that are not in a relation with each other, but would both prefer each other over their current partners. Though stability in this context definitely seems to be a desirable property, we show in Chapter 2 that it cannot be obtained for free but comes only at the price of a significantly reduced overall happiness of the underlying society.

In the second part of this thesis, we focus on another, to some extent more futuristic, aspect of collaboration: the realm of *autonomous robots*. One of the most aspiring goals of robotics scientists is the design and construction of large numbers of inexpensive robots that are able to collaboratively and autonomously accomplish complex tasks. As of today, robots are already able to play soccer in teams (see, e.g., [69, 70, 90, 94]), fly and dance autonomously in close vicinity without crashing into each other (see, e.g., [16, 17]), self-assemble complex structures with hundreds to thousands of units (see, e.g., [93, 99]), etc. In Chapter 3, we consider the problem of robots having to gather in one point of the plane — a special case of the structure-assembly problem — from a theoretical perspective. The robots have very limited capabilities and in particular cannot communicate other than observing each other positions within a limited viewing range. We analyze an algorithm from a paper from 1999 and prove a long-standing conjecture claiming that the robots gather within polynomially many rounds. More specifically, we show that the robots gather in $\mathcal{O}(n^2)$ rounds and also give a matching lower bound.

Interpreting autonomous robots in a more natural, i.e., biological sense, one quickly enters the field of *swarm intelligence*. Collins dictionary gives, among others, the following definition: "the collective behavior of a group of animals, esp. social insects such as ants, bees, and termites, that are each following very basic rules". Most of us have come in contact with collaborative behavior of insects in secondary school, when learning for example about the intricate tail-wagging dance that bees use to communicate the direction and distance to the next food source. Social insects have developed a breath-taking variety of techniques to accomplish many complex goals such as foraging food, building nests, efficiently allocating different tasks to different types of workers, evaluating different nesting places, etc. It is important to realize that for most insects, collaboration is not just a means to increase the output of the colony, but an essential requirement to perform certain tasks correctly.

Sasaki et al. [100] recently studied ant colonies of the species *Temnothorax rugatulus*, which frequently have to find new nest sites. When confronted with only two different nest sites of different quality, individuals are very accurate (approx. 90%) in finding the better option. When the number of sites is increased to eight while maintaining the same ratio of good vs. bad sites, the individual ant suffers from a sort of cognitive overload and therefore its accuracy deteriorates to that of a random choice (approx. 50%). The colony, however, does not appear

to suffer from this deficiency, as their collaborative feedback mechanism ensures a accuracy of around 90%, even in the more difficult case with many options.

There is a long list of many more examples of collaboration in the animal (and plant) kingdom, which shows that it is a key concept responsible for the survival (and co-existence) of many species on our planet. In the second part of the thesis, we aim to advance our understanding of the theoretical aspects of collaboration among mobile agents ("ants") with limited capabilities. In Chapter 4, we model the process of a colony of ants searching for food formally, and show that the ants are able to efficiently locate a food source on a two-dimensional integer grid even when the individuals are controlled only by finite state machines and operate in an asynchronous environment. In Chapter 5 we look at small numbers of ants and establish upper and lower bounds on the number of ants required to successfully and efficiently locate the food source for several variants of the model.

In the third and last part of this thesis, we undertake a brief excursion into the darker aspects of collaboration: *collusion*. Merriam-Webster defines the word as "secret agreement or cooperation especially for an illegal or deceitful purpose". In Chapter 6, we give evidence towards the fact that collaboration is not always a means towards improving the welfare of a society, but can also be quite detrimental to it. We describe an attack against an online poker platform, exploiting the weakness of such platforms to effectively control the configuration of players at a certain table. We developed a system of automated players (bots) that join the same table and exchange information about their hands among each other in order to promote their winning probabilities. Our experiments on play money tables show that the performance of the bots scales almost linearly with the number of colluding players.

1.1 Notational Conventions

In this thesis we mostly use the standardized notation that has been established in our field over the previous decades and centuries. A few notations and definitions, however, are not entirely unambiguous, which is why we specify their exact meaning in this section.

1.1.1 Special Sets

We denote the base sets $\mathbb{Z}$, $\mathbb{Q}$, and $\mathbb{R}$ as the sets of all integers, rational numbers, and real numbers, respectively. When suitable, we further restrict these sets by adding a superscript $+$ to denote only the positive elements (e.g., the positive reals $\mathbb{R}^+$) and further add a subscript 0 if we want to include the element 0 (e.g., the natural numbers $\mathbb{Z}_0^+$).

For an arbitrary set S, we denote by 2^S the *power set* of S containing all subsets of S including the empty set and S itself.

1.1.2 Bachmann-Landau Notations

For two functions f and g defined on the non-negative integers $\mathbb{Z}_0^+$, we use the Bachmann-Landau notations in the following sense. We write

- $f \in o(g)$ iff f grows asymptotically slower than g, or formally

$$\lim_{n\to\infty} \left|\frac{f(n)}{g(n)}\right| = 0 \ ,$$

- $f \in \mathcal{O}(g)$ iff f does not grow asymptotically faster than g, or formally

$$\limsup_{n\to\infty} \left|\frac{f(n)}{g(n)}\right| < \infty \ ,$$

- $f \in \Theta(g)$ iff f grows asymptotically as fast as g, or formally

$$0 < \liminf_{x\to\infty} \left|\frac{f(x)}{g(x)}\right| \leq \limsup_{x\to\infty} \left|\frac{f(x)}{g(x)}\right| < \infty \ ,$$

- $f \in \Omega(g)$ iff f does not grow asymptotically slower than g, or formally

$$\liminf_{n\to\infty} \left|\frac{f(n)}{g(n)}\right| > 0 \ ,$$

- $f \in \omega(g)$ iff f grows asymptotically faster than g, or formally

$$\lim_{n\to\infty} \left|\frac{f(n)}{g(n)}\right| = \infty \ .$$

We sometimes abuse notation and write $f = \diamond(n)$ instead of $f \in \diamond(n)$ for $\diamond \in \{o, \mathcal{O}, \Theta, \Omega, \omega\}$. With the expression $f(n) = h(n) + \mathcal{O}(g(n))$, for example, we want to say that there exists a constant c such that $f(n) \leq h(n) + c \cdot g(n)$ for sufficiently large n.

1.1.3 Miscellaneous

We say that an event occurs *with high probability* with respect to n (usually the problem size), abbreviated by w.h.p., if the event occurs with probability at least $1 - n^{-c}$, where c is an arbitrarily large constant.

When we write $\log x$ without specifying a base, we denote the logarithm of x to the base of 2.

1.2 Collaborations and Contributions

This thesis is based on several publications and drafts I worked on during my time as a PhD student at the Distributed Computing group at ETH Zurich under the supervision of Prof. Dr. Roger Wattenhofer. As it is quite often the nature of research, I did not do all the work myself but collaborated with other

brilliant people. I shall list in the following the publications or drafts underlying the respective chapters along with a list of the respective co-authors. Note that the authors of the papers are listed in alphabetical order and therefore do not represent the degree of contribution of the individual authors. Besides this, all articles listed below are joint work with my supervisor Prof. Dr. Roger Wattenhofer.

Chapter 2 is based on the publication draft *The Price of Matching with Metric Preferences*. Co-author was Yuval Emek.

Chapter 3 is based on the publication *A Tight Runtime Bound for Synchronous Gathering of Autonomous Robots with Limited Visibility* [39]. Co-authors were Bastian Degener, Barbara Kempkes, Friedhelm Meyer auf der Heide, and Peter Pietrzyk.

Chapter 4 is based on the publication *Solving the ANTS Problem with Asynchronous Finite State Machines* [50]. Co-authors were Yuval Emek and Jara Uitto.

Chapter 5 is based on the publication *How Many Ants Does It Take To Find the Food?* [49]. Co-authors were Yuval Emek, David Stolz, and Jara Uitto.

Chapter 6 is based on Benjamin Zehnder's Bachelor thesis [111], that I supervised in the spring 2012. Co-authors were Thomas Locher and Benjamin Zehnder.

Part I

Stable Matching

2

Stable Matching in Metric Graphs

Medical students, players of online games, organ receivers, paper reviewers, users of dating websites, and many others have in common that they are being assigned to their partners by a match-making process. Indeed, matching can be found in many settings, and is one of the core concepts in society and economics. Depending on the circumstances, the matching process can be centralized, with a central authority deciding on who is matched to whom, more anarchic, with the participants deciding on their matches themselves, or something in between, with a neutral advisor helping the participants to find good matches.

In this chapter, we aim to connect two classic approaches towards *matching*. The first approach tackles matching from a (global) optimization angle à la Edmonds [47, 48]: given $2n$ nodes with pairwise costs $c(x, y) = c(y, x) \in \mathbb{R}_{>0}$, the goal is to construct a perfect matching that minimizes the total cost. The second approach tackles matching from the (local) selfish angle à la Gale and Shapley [59]: each node is equipped with a preference list ranking its potential matches and a matching is *stable* if no two unmatched nodes prefer each other over their current matches.

We consider a restricted case of the stable matching realm, where the nodes preferences are determined based on the aforementioned pairwise costs $c(\cdot, \cdot)$ so that node x prefers node y over node y' if and only if $c(x, y) < c(x, y')$, and focus on the following question: How does the requirement that the returned matching has to be (locally) stable affect its (global) total cost? In an attempt to provide a quantitative answer to this question, we shall look at matching instances through the *Price of Stability (PoS)* lens that compares the min-cost stable matching to the unrestricted optimum, measuring the ratio of their respective costs. In

fact, to provide a deeper understanding of the delicate balance between the global matching cost and its local stability, we generalize the problem by using the notion of an α-*stable* matching for $\alpha \geq 1$, in which no pair of unmatched nodes has an incentive to defect in order to improve both their costs by a factor (strictly) greater than α.

Unfortunately, in general, the Price of Stability may be unbounded, as the following simple example shows: Let G be a complete graph on four nodes u_1, u_2, v_1, v_2 with edge costs $c(u_1, v_1) = c(u_2, v_2) = 1$, $c(u_1, u_2) = \varepsilon$ for some small $\varepsilon > 0$, and $c(v_1, v_2) = c(u_1, v_2) = c(u_2, v_1) = C$ for some large C. Then, the optimal perfect matching matches u_i to v_i for $i = 1, 2$ with a cost of 2, whereas an α-stable matching for any reasonable value of α must match u_1 to u_2, and hence also v_1 to v_2 which incurs an arbitrarily large cost.

Fortunately, real-world matching instances often exhibit *metric* costs, i.e., costs that satisfy the triangle inequality (or its bipartite counterpart). Examples include assigning medical students to hospitals, organ receivers to donors, papers to reviewers, and also players in online games, or users of dating websites. While the metric character of the first few examples is intuitive, appreciating the metric character of online dating is slightly more demanding — refer to the next section for a comprehensive explanation that also addresses the role that PoS plays in these matching scenarios.

Indeed, we establish an upper bound of $\mathcal{O}(n^{\log(3/2)})$ on the PoA and PoS of minimum-cost perfect matchings in metric graphs with $2n$ vertices, where $\log(3/2) \simeq 0.58$, and show that this is asymptotically tight. The somewhat unattractive polynomial dependency on n raises the following question: How does PoS improve once the Gale-Shapley stability is relaxed to α-*stability*, where two unmatched vertices deviate from the current matching only if both improve their costs by a factor greater than $\alpha \geq 1$? (Observe that since, by definition, every stable matching is also α-stable, this question is irrelevant in the context of PoA that can only increase by such a relaxation.) We answer this question by establishing an asymptotically tight trade-off, showing that with respect to α-stable matchings, PoS improves to $\Theta(n^{\log(1+1/(2\alpha))})$; in particular, taking $\alpha = \mathcal{O}(\log n)$ yields a constant PoS. All our results hold for both simple and bipartite metric graphs.

Online Dating. Consider a (heterosexual) dating platform with the goal to help a set W of women and a set M of men finding suitable partners and let us assume that $|W| = |M|$. When signing up for the platform, a woman $w \in W$ (resp., a man $m \in M$) has to complete a questionnaire about a set of her (resp., his) own characteristics including absolute numerical properties such as age, height, or weight; relative numerical properties such as sportiness, empathy, or ambition; and combinations thereof that capture more complex aspects such as movie taste. This questionnaire generates a point $v(w)$ (resp., $v(m)$) in the real vector space V_W (resp., V_M), where each characteristic is represented by one or more dimensions. Then, she (resp., he) completes a different questionnaire describing the characteristics of her ideal male partner w^* (resp., his ideal female partner m^*), which in turn generates a point $v(w^*)$ in V_M (resp., $v(m^*)$ in V_W).

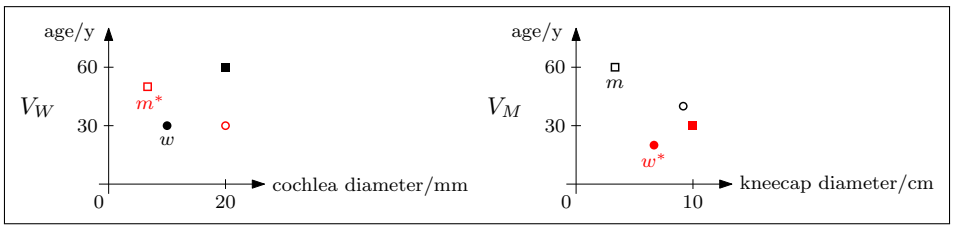


Figure 2.1: In a fictitious example where age is relevant for both sexes while men are particularly interested in women's cochleas and women care about men's kneecaps, we depict the points corresponding to the characteristics of four individuals, two women and two men. A black mark indicates the characteristic of a specific user while the corresponding shape in red marks the characteristic of that person's ideal partner.

Notice that V_M and V_W may correspond to different sets of characteristics (after all, men and women might be interested in different qualities of the respective opposite sex).

We assume that the valuation of woman w for man m is determined by $\|v(m), v(w^*)\|_M$, where $\|\cdot\|_M$ is some norm on V_M. (The same goes with the opposite sexes with respect to some norm $\|\cdot\|_W$ on V_W.) Thus, the cost of a match between a woman $w \in W$ and a man $m \in M$ can be measured in terms of $c(w,m) = \|v(m), v(w^*)\|_M + \|v(w), v(m^*)\|_W$ with the reasoning that the smaller the cost $c(w,m)$ is, the better w and m fit together. Observe that by definition, the cost function $c(\cdot,\cdot)$ obeys the bipartite counterpart of the triangle inequality. Refer to Figure 2.1 for an example with two-dimensional vector spaces V_W and V_M.

The popularity of a dating platform depends significantly on the trust that its users have in the platform finding them a suitable partner. This trust can be boosted by providing rigid guarantees for the matching established by the platform. Two natural such guarantees are that the total matching cost is minimized, which means that in total, the happiness among the participants is maximized; and that the matching is stable, which means that no unmatched pair has an incentive to deviate from the matching recommended by the platform. However, it turns out that these two guarantees cannot coexist, hence we allow for an approximation of the minimum cost (perfect) matching and relax the notion of stability to α-stability (which will be defined precisely soon). The PoS then tells us how well the dating platform can do in terms of the total matching cost compared to a benchmark which is not subject to the (relaxed) stability constraint.

The problem described above corresponds to finding matchings in complete bipartite graphs (the marriage version). If, instead, we consider a same-sex dating platform, we end up with points in a single vector space and a cost function defined over all user pairs which corresponds to matchings in complete graphs (the roommates version).

2.1 Related Work

Studying the impact of selfish players has been a major theoretical computer science success story in the last decade (see, e.g., the 2012 *Gödel Prize* [75,83,98]). In particular, much effort has been invested in quantifying how the efficiency of a system degrades due to selfishness of its players. The most notable notions in this context are the *Price of Anarchy (PoA)* [75, 85] and the *Price of Stability (PoS)* [12, 101], comparing the best possible outcome to the outcome of the worst (PoA) or best (PoS) solution with selfish players. Since their introduction, the Price of Anarchy and the Price of Stability have been extensively analyzed in diverse settings such as selfish routing [12,18,30,31,97,98,105], network formation games [3,8,13,29,108], job scheduling [19,36,74,75], and resource allocation [68,96]. While selfish players are traditionally modeled using the *Nash equilibrium* solution concept, where no player can benefit from a unilateral deviation, in matching settings unilateral deviations are not natural. Instead, we want that no *two* unmatched players prefer each other over their current matching partners. This solution concept is generally known as the Gale-Shapley *stable matching* [59].

For the most part, the stable matching realm has been subdivided into two versions: the *marriage* (bipartite) version, where the players are partitioned into *men* and *women* and each man (resp., woman) is equipped with a list of preferences over the set of women (resp., men); and the *roommates* (all-pairs) version, where each player is equipped with a list of preferences over all other players. Gale and Shapley showed that in the bipartite version, a stable matching always exists, and in fact, can be computed by a simple poly-time algorithm. In contrast, the all-pairs version does not necessarily have a solution. Both versions of the stable matching problem and their manifold variants (strictly/weakly ordered preferences, (in-)complete preference lists, (a-)symmetric preferences) admit an abundance of literature; see, e.g., the books of Knuth [73], Gusfield and Irving [61], and Roth and Sotomayor [95]. The notion of stability studied in this chapter has been coined as *strong stability* by Irving [63].

Sometimes, the players' preferences are associated with real costs so that each preference list is sorted in order of non-decreasing costs. This setting gives rise to the *minimum-cost stable matching* problem, where the goal is to construct a stable matching that minimizes the total cost of matched partners. Irving et al. [64] designed a poly-time algorithm for the bipartite (marriage) version of a special case of this problem, referred to as the *egalitarian stable matching* problem, where a cost of j is associated with each player for matching his/her jth preferred partner. This was generalized by Feder [52] who presented a poly-time algorithm for the bipartite version of the general minimum-cost stable matching problem. Moreover, Feder also established the NP-hardness of the all-pairs version and showed that it admits a 2-approximation algorithm.

The players' preferences in general stable matching scenarios exhibit no intrinsic correlations. Several approaches have been taken towards introducing consistency in the preference lists [65, 73, 82]. Most relevant to the current chapter is the approach of Arkin et al. [14] who studied the *geometric* stable

roommate problem, where the players are identified with points in a Euclidean space and the preferences are given by the sorted distances to the other points. They showed that in the geometric setting, a stable matching always exists and that it is unique if the players' preferences exhibit no ties. These results easily generalize to arbitrary metric spaces. Arkin et al. also introduced the notion of an *α-stable* matching, which is central to the current chapter.

Reingold and Tarjan [88] proved that the approximation ratio of some greedy algorithm for minimum-cost perfect matching in metric graphs is $\Theta(n^{\log(3/2)})$ where $\log(3/2) \approx 0.58$. We point out that this result is equivalent to establishing the same bound for the PoA of minimum-cost perfect matching in such graphs. In Section 2.3 we give a simpler proof for the PoA-result and extend their result in Section 2.4 to obtain a lower bound for the PoS for all $\alpha \geq 1$ of $\Omega(n^{\log(1+1/(2\alpha))})$.

2.2 Setting and Preliminaries

Consider a graph G with vertex set $V(G)$ and edge set $E(G)$. Each edge $e \in E(G)$ is assigned a positive real *cost* $c(e)$. Unless stated otherwise, the graphs mentioned have $2n$ vertices, $n \in \mathbb{Z}^+$, and are either complete ($|E(G)| = \binom{2n}{2}$) or complete bipartite ($V(G) = U_1 \cup U_2$, $|U_1| = |U_2| = n$ and $|E(G)| = n^2$). We say that the complete graph G is *metric* if $c(x,y) \leq c(x,z) + c(z,y)$ for every $x, y, z \in V(G)$; we say that the complete bipartite graph G is *metric* if $c(x,y) \leq c(x,z) + c(z,z') + c(z',y)$ for every $x, y, z, z' \in V(G)$, where x, z' and y, z are on opposite sides of the bipartite graph. For an arbitrary graph G, the *distance* $\text{dist}_G(x,y)$ of two vertices x and y of G is defined as the weighted length of the shortest path between x and y in G.

A *matching* is a subset $M \subseteq E(G)$ of the edges such that every vertex in $V(G)$ is incident to at most one edge in M. A matching is called *perfect* if every vertex in $V(G)$ is incident to exactly one edge in M, which implies that $|M| = n$ as $|V(G)| = 2n$. For a perfect matching M and a vertex $x \in V(G)$, we denote by $M(x)$ the unique vertex $y \in V(G)$ such that $(x,y) \in M$. Unless stated otherwise, all matchings mentioned hereafter are assumed to be perfect. (Perfect matchings clearly exist in a complete or complete balanced bipartite graph with an even number of vertices.) Given an edge subset $F \subseteq E(G)$, we define the *cost* of F as the total cost of all edges in F, denoted by $c(F) = \sum_{e \in F} c(e)$; in particular, the cost of a matching is the sum of its edge costs.

Definition (α-Stable Matching)**.** Consider some (perfect) matching $M \subseteq E(G)$ and some real number $\alpha \geq 1$. An edge $(u,v) \notin M$ is called *α-unstable* with respect to M if $\alpha \cdot c(u,v) < \min\{c(u, M(u)), c(v, M(v))\}$. Otherwise, the edge is called *α-stable*. A matching M is called *α-stable* if it does not admit any α-unstable edge. We will omit α and call edges as well as matchings just *stable* or *unstable* whenever α is clear from the context or the argumentation holds for every choice of α.

Let M^* denote a certain (perfect) matching M that minimizes $c(M)$. For simplicity, we restrict our attention to complete (rather than complete bipartite) metric graphs, although all our results hold also for the complete bipartite case.

Definition (α-Price of Stability)**.** The *α-Price of Stability* of G, denoted by $\mathrm{PoS}_\alpha(G)$, is defined as $\mathrm{PoS}_\alpha(G) = \min\{c(M)/c(M^*) : M \text{ is } \alpha\text{-stable matching}\}$. Furthermore, $\mathrm{PoS}_\alpha(2n) = \sup\{\mathrm{PoS}_\alpha(G) : G \text{ is metric}, |V(G)| = 2n\}$. Unless stated otherwise, when the parameter α is omitted, we refer to the case $\alpha = 1$.

Definition (Price of Anarchy)**.** The *Price of Anarchy* of a graph G, denoted by $\mathrm{PoA}(G)$, is defined as $\mathrm{PoA}(G) = \max\{c(M)/c(M^*) : M \text{ is stable matching}\}$. Furthermore, $\mathrm{PoA}(2n) = \sup\{\mathrm{PoA}(G) : G \text{ is metric}, |V(G)| = 2n\}$.

Note that since any stable matching by definition is also α-stable for any $\alpha \geq 1$, the Price of Anarchy does not improve by considering α-stability and hence its definition does not include the parameter α.

2.3 Price of Anarchy

The following theorem was implicitly proven by Reingold and Tarjan [88] already in 1981. They showed that for minimum-cost perfect matching in metric graphs, the approximation ratio of the algorithm that picks edges by ascending costs is $\Theta(n^{\log(3/2)})$. Since the matching returned by this greedy algorithm is stable and since every stable matching can be obtained from the algorithm by an appropriate tie-breaking policy, it follows that the PoA of minimum-cost perfect matching in such graphs is also $\Theta(n^{\log(3/2)})$.

Theorem 2.1. *The PoA of minimum-cost perfect matching in metric graphs with $2n$ vertices is $\Theta(n^{\log(3/2)})$.*

We present a simpler and more intuitive proof for Reingold and Tarjan's 30-year-old result, which essentially relies on a series of elementary reductions, essentially showing that $\mathrm{PoA}(2n)$ is realized by weighted line graphs, i.e., metric graphs that can be embedded isometrically into the real line. Following that, we introduce a family of weighted line graphs with PoA of $\Theta(n^{\log(3/2)})$ and show that no other weighted line graph admits higher PoA.

Definition (Matching Configuration)**.** A *matching configuration (MC)* $\xi = (G, M^*, M)$ consists of a metric graph G, a minimum-cost matching M^*, and a stable matching M on G. The *ratio* of ξ is defined as $\rho(\xi) := c(M)/c(M^*)$.

Observe that the definition of a MC ξ induces a collection $\mathcal{A}(\xi)$ of alternating cycles in the symmetric difference $M \oplus M^*$, where an alternating cycle is a cycle whose edges are alternatingly from M and M^*. The cycles in $\mathcal{A}(\xi)$ are referred to hereafter as the alternating cycles *exhibited* by ξ. We say that ξ is *spanned* by the cycles in $\mathcal{A}(\xi)$ if each vertex of G belongs to an alternating cycle in $\mathcal{A}(\xi)$. Clearly, graphs with two vertices admit a single (perfect) matching, hence $\mathrm{PoA}(2) = 1$, so in what follows, it suffices to consider MCs on $2n$ vertices for $n > 1$. The following lemma states that it also suffices to consider MCs spanned by a single alternating cycle.

Lemma 2.1. *For every MC $\xi = (G, M^*, M)$ on $2n$ vertices, there exists a MC $\hat{\xi}$ on $2n'$ vertices, $1 < n' \leq n$, spanned by a single alternating cycle such that $\rho(\hat{\xi}) \geq \rho(\xi)$.*

Proof. Since $\mathcal{A}(\xi) = \emptyset$ implies $\rho(\xi) = 1$, we may assume hereafter that $|\mathcal{A}(\xi)| \geq 1$. So let A be an alternating cycle in $\mathcal{A}(\xi)$ maximizing the ratio $c(M_A)/c(M_A^*)$, where M_A and M_A^* are the matchings M^* and M, resp., restricted to the edges of A. Let G_A be the subgraph of G induced by $V(A)$ and take $\hat{\xi} = (G_A, M_A^*, M_A)$. Observe that $\hat{\xi}$ is a valid MC, since M_A^* and M_A are still a minimum-cost matching and a stable matching, resp., in G_A. By the choice of A, it follows that $\rho(\hat{\xi}) \geq \rho(\xi)$. □

In the following, we will say that the edge costs in a graph $G = (V, E)$ *agree* with the distances in a subgraph $G' = (V, E')$ on the same vertices, iff for any edge (x, y) in G we have $c(x, y) = \text{dist}_{G'}(x, y)$.

Definition (Weighted Cycle MC)**.** A MC $\xi = (G, M^*, M)$ is said to be a *weighted cycle MC* if ξ is spanned by a single alternating cycle A and the edge costs in G agree with the distances in the subgraph of G induced by the edges in $E(A)$.

Our next lemma states that it suffices to bound the PoA in weighted cycle MCs.

Lemma 2.2. *For every MC $\xi = (G, M^*, M)$ on $2n$ vertices that is spanned by a single alternating cycle, there exists a weighted cycle MC $\hat{\xi}$ on $2n$ vertices such that $\rho(\hat{\xi}) \geq \rho(\xi)$.*

Proof. Let A be the single alternating cycle spanning ξ. If ξ is not a weighted cycle MC, then G must admit a *shortcut* — an edge $(x, y) \in E(G) \setminus E(A)$ satisfying $c(x, y) < \text{dist}_A(x, y)$, where $\text{dist}_A(x, y)$ denotes the distance between x and y in the (weighted) cycle A. Let (x, y) be a shortcut minimizing $c(x, y)$ and let $z \in V(G) \setminus \{x, y\}$ be the vertex minimizing $c(x, z) + c(z, y)$. Observe that $c(x, y)$ must be strictly smaller than $c(x, z) + c(z, y)$ as (x, y) is a shortcut of G and G does not admit any shorter shortcut. We argue that the cost of (x, y) can be increased to $c(x, z) + c(z, y)$ without violating the validity of ξ as a MC. As there are only finitely many shortcuts, the assertion follows since repeating this step (finitely many times) removes all the shortcuts of G. To that end, note that after increasing $c(x, y)$ to $c(x, z) + c(z, y)$, M^* remains a minimum-cost matching of G (we only increased the cost of some edge not in M^*) and M remains a stable matching of G (we only increased the cost of some edge not in M). So, all we have to show is that G remains metric, which follows from the choice of z. □

Definition (Weighted Line MC)**.** We say that a $(2n)$-vertex metric graph G is a *weighted line graph* if it can be isometrically embedded into the real line. As such, it is convenient to identify the vertices of G with the reals $x_1 < \cdots < x_{2n}$ so that $c(x_i, x_j) = x_j - x_i$ for every $1 \leq i < j \leq 2n$. In some cases, it will also be convenient to define a weighted line graph by setting all the differences $x_{i+1} - x_i$ without explicitly specifying the x_is themselves. A *weighted line MC* $\xi = (G, M^*, M)$ is a MC on $2n$ vertices satisfying:

(1) G is a weighted line graph;

(2) $M^* = \{(x_{2i-1}, x_{2i}) \mid 1 \leq i \leq n\}$; and

(3) $M = \{(x_{2i}, x_{2i+1}) \mid 1 \leq i < n\} \cup \{(x_1, x_{2n})\}$.

Observe that ξ is spanned by a single alternating cycle $A = (x_1, \ldots, x_{2n}, x_1)$.

Note that requirement (2) in the definition is not really necessary: the requirement that G is a weighted line graph already implies that $\{(x_{2i-1}, x_{2i}) \mid 1 \leq i \leq n\}$ is the unique minimum-cost matching of G as every other matching M' contains some edge (x_i, x_j) such that $|j - i| > 1$; it is easy to show that such an edge must belong to an improving alternating cycle, hence M' cannot be optimal. Given a $(2n)$-vertex weighted line graph G, we shall subsequently denote this unique minimum-cost stable matching by $M^*(G)$ and the matching $\{(x_{2i}, x_{2i+1}) \mid 1 \leq i < n\} \cup \{(x_1, x_{2n})\}$ by $M(G)$. By definition, $\xi = (G, M^*(G), M(G))$ is a valid (weighted line) MC if and only if $M(G)$ is stable. Note also that a weighted line MC is a refinement of a weighted cycle MC, with the additional requirement that the cost of the longest edge in the unique alternating cycle A equals the total cost of all other edges of A. Building on this fact, the next lemma states that it suffices to consider weighted line MCs.

Lemma 2.3. *For every weighted cycle MC $\xi = (G, M^*, M)$ on $2n$ vertices, there exists a weighted line MC $\hat{\xi}$ on $2n$ vertices such that $\rho(\hat{\xi}) \geq \rho(\xi)$.*

Proof. Let A be the single alternating cycle spanning ξ and let e be an edge in M that maximizes $c(e)$. Let $W_{-e} = \sum_{e' \in E(A) \setminus \{e\}} c(e')$. Clearly, $c(e) \leq W_{-e}$, as otherwise, G is not metric. We argue that if $c(e) < W_{-e}$, then the cost of e can be increased to W_{-e} (while also adapting the cost of all edges whose cost is affected by e) without violating the validity of ξ as a MC; the assertion follows because this step turns ξ into a weighted line MC. To that end, note that after increasing $c(e)$ to W_{-e}, G remains metric (ξ is a weighted cycle MC) and M^* remains a minimum-cost matching (we only increased the cost of some edges not in M^*). So, all we have to show is that M remains stable, which follows from the choice of e. □

Once we restrict our attention to weighted line configurations, we can augment G with new vertices without significantly affecting the ratio of the MC.

Lemma 2.4. *For every weighted line MC $\xi = (G, M^*, M)$ on $2n$ vertices and for any $\varepsilon > 0$, there exists a weighted line MC $\hat{\xi}$ on $2(n+1)$ vertices such that $\rho(\hat{\xi}) \geq \rho(\xi) - \varepsilon$.*

Proof. Recall that the vertices of G are identified with the reals $x_1 < \ldots < x_{2n}$. Let $\hat{G}$ be the weighted line graph obtained from G by augmenting $V(G)$ with two new vertices identified with the reals $y = x_{2n} + \delta$ ands $y' = y + \delta'$ for some $\delta' > \delta > 0$ (see Figure 2.2 for an illustration). The assertion follows since by taking a sufficiently small δ', we guarantee both that $M(\hat{G})$ is stable in $\hat{G}$ and that $c(M(\hat{G}))/c(M^*(\hat{G})) \geq \rho(\xi) - \varepsilon$. □

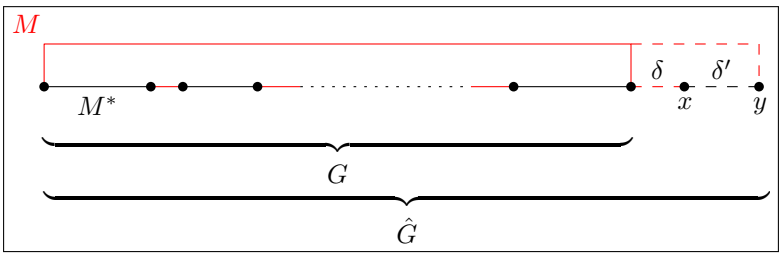


Figure 2.2: A weighted line MC can be augmented with two vertices x and y in distances δ and δ' without changing the structure of its matchings and without affecting its ratio ρ by more than ε. The red edges depict the stable matching M while the black edges depict the minimum-cost matching M^*. The dashed edges show the changes of the two matchings after the augmentation.

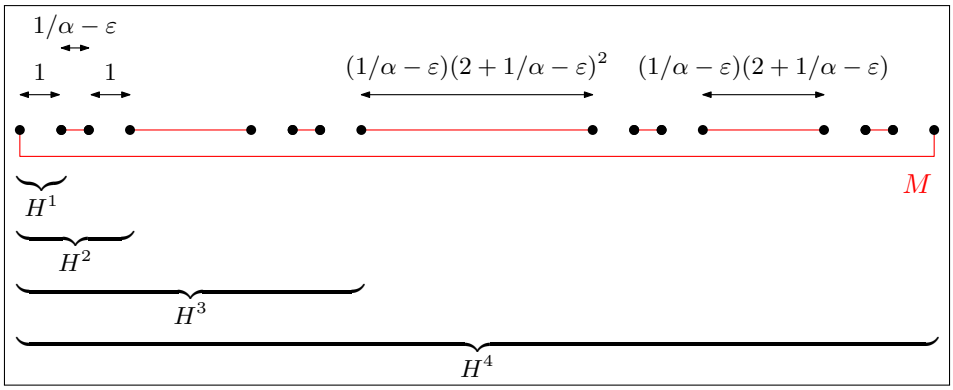


Figure 2.3: This parametrized Reingold-Tarjan graph H^4_α with 2^4 vertices has a unique "expensive" α-stable matching M (red edges). Setting the optional parameters α and ε (that are used in the proof of the PoS lower bound) to 1 and 0, resp., yields the original Reingold-Tarjan graph H^4.

We now turn to present a family of metric graphs referred to as *Reingold-Tarjan graphs*, acknowledging Reingold and Tarjan's paper [88], where these graphs were first introduced.

Consider some integer $k > 0$. The k^{th} Reingold-Tarjan graph H^k is a weighted line graph whose 2^k vertices are identified with the reals $x^k_1 < \cdots < x^k_{2^k}$. It is defined recursively: For $k = 1$, we set $x^1_2 - x^1_1 = 1$. Assume that H^k is already defined and let $D^k = x^k_{2^k} - x^k_1$ be its *diameter*. Then, H^{k+1} is defined by placing two disjoint instances of H^k on the real line with an S^{k+1} *spacing* between them, i.e., $x^{k+1}_{2^k+1} - x^{k+1}_{2^k} = S^{k+1}$, yielding $D^{k+1} = 2 \cdot D^k + S^{k+1}$. In the current[1] construction, we set $S^k = D^{k-1}$, thus the diameter of H^k satisfies $D^k = 3^{k-1}$. Refer to Figure 2.3 for an illustration of the parametrized Reingold-Tarjan graph H^4_α that will be used later. The original graph is obtained by setting the two parameters α and ϵ to 1 and 0, respectively.

Recall that $M^*(H^k)$ matches x^k_{2i-1} with x^k_{2i} for every $1 \le i \le 2^{k-1}$; since

[1] A parametrized variant of the Reingold-Tarjan graphs is presented in Section 2.4, where we use a different value for S^k.

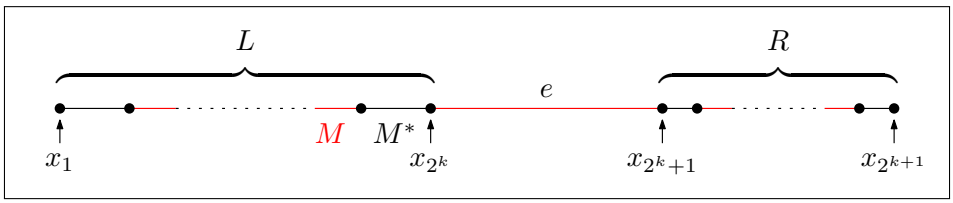


Figure 2.4: Any MC ξ on 2^k vertices can be transformed into a Reingold-Tarjan MC without decreasing the ratio $\rho(\xi)$. The black edges are part of the minimum-cost matching M^* while the red edges belong to the stable matching M.

all these edges have cost 1, it follows that $c(M^*(H^k)) = 2^{k-1}$. Furthermore, we argue by induction on k that the matching

$$M(H^k) = \{(x^k_{2i}, x^k_{2i+1}) \mid 1 \leq i < 2^{k-1}\} \cup \{(x^k_1, x^k_{2^k})\}$$

is stable and that its cost is

$$c(M(H^k)) = D^k + (D^k - c(M^*)) = 2 \cdot 3^{k-1} - 2^{k-1} \ .$$

Therefore, $\xi^k_{RT} = (H^k, M^*(H^k), M(H^k))$, referred to as the k^{th} *Reingold-Tarjan MC* hereafter, is a valid weighted line MC with ratio

$$\rho(\xi^k_{RT}) \ = \ \frac{c(M(H^k))}{c(M^*(H^k))} \ = \ \frac{2 \cdot 3^{k-1} - 2^{k-1}}{2^{k-1}} \ = \ \Theta((3/2)^{k-1}) \ = \ \Theta(n^{\log(3/2)}) \ ,$$

where the last equation follows by setting $2n = 2^k$. Combined with Lemma 2.4, we immediately conclude that $\text{PoA}(2n) = \Omega(n^{\log(3/2)})$, establishing the lower bound part of Theorem 2.1. The upper bound part of the theorem is established by combining Lemmas 2.1, 2.2, 2.3, and 2.4 with the following lemma.

Lemma 2.5. *The k^{th} Reingold-Tarjan MC ξ^k_{RT} satisfies the inequality $\rho(\xi^k_{RT}) \geq \rho(\xi)$ for any weighted line MC ξ on 2^k vertices.*

Proof. By induction on k. The assertion holds trivially for $k = 1$, so assume that it holds for k and consider an arbitrary weighted line MC $\xi = (G, M^*(G), M(G))$ on 2^{k+1} vertices identified with the reals $x_1 < \cdots < x_{2^{k+1}}$. Let L and R be the subgraphs of G induced by the vertices $x_1, \ldots, x_{2^k}$ and $x_{2^k+1}, \ldots, x_{2^{k+1}}$, resp. Let $e = (x_{2^k}, x_{2^k+1})$ and let $D_L = x_{2^k} - x_1$ and $D_R = x_{2^{k+1}} - x_{2^k+1}$. We refer to the vertices x_1 and x_{2^k} (resp., x_{2^k+1} and $x_{2^{k+1}}$) as the *external* vertices of L (resp., R) and to the vertices $x_2, \ldots, x_{2^k-1}$ (resp., $x_{2^k+2}, \ldots, x_{2^{k+1}-1}$) as the *internal* vertices of L (resp., R). Observe that $e \in M(G)$ and since $M(G)$ is a stable matching of G, we must have $x_{2^k+1} - x_{2^k} = c(e) \leq \min\{D_L, D_R\}$ as otherwise, at least one of the edges (x_1, x_{2^k}) or $(x_{2^k+1}, x_{2^{k+1}})$ is unstable. Figure 2.4 illustrates the various notions.

We say that a 2^k-vertex weighted line graph is *consistent* with H^k if it can be obtained from H^k by scaling the edge costs. Fixing the external vertices of L and R, we argue that the internal vertices of L and R can be repositioned so that L and R, resp., become consistent with H^k without violating the validity of ξ as a

weighted line MC and without decreasing the ratio $\rho(\xi)$. We shall establish this fact for L; the proof for R is analogous. Note first that since $M(H^k)$ is stable in H^k and since $c(e) \leq D_L$, it follows that by repositioning the internal vertices of L so that L becomes consistent with H^k, we do not violate the stability of $M(G)$. Second, by the inductive hypothesis, repositioning the internal vertices of L so that L becomes consistent with H^k maximizes $c(M(L))/c(M^*(L))$, thus $\rho(\xi)$ cannot decrease after this repositioning step, which establishes the argument. So, assume hereafter that both L and R are consistent with H^k.

Assume without loss of generality that $D_L \geq D_R$, so $c(e) = x_{2^k+1} - x_{2^k}$ is at most D_R. In fact, since R is consistent with H^k, it follows that we can increase the difference $x_{2^k+1} - x_{2^k}$ until it is equal to D_R, keeping the difference $x_{i+1} - x_i$ unchanged for all other is, without violating the validity of ξ as a weighted line MC and without decreasing the ratio $\rho(\xi)$. So, assume hereafter that $D_L \geq c(e) = D_R$. Now, we argue that we can scale down the differences $x_{i+1} - x_i$ for every $1 \leq i < 2^k$, keeping $x_{i+1} - x_i$ unchanged for all other is, until we obtain $D_L = c(e) = D_R$, without decreasing the ratio $\rho(\xi)$. This completes the proof since $D_L = c(e) = D_R$ implies that $G = H^{k+1}$.

Let $\ell = c(M(L)) - D_L$, $\ell^* = c(M^*(L))$, $r = c(M(R)) - D_R$, and $r^* = c(M^*(R))$; notice that $\ell + \ell^* = D_L$ and $r + r^* = D_R$. Since $c(e) = D_R$, we can express $\rho(\xi)$ as

$$\rho(\xi) = \frac{c(M(G))}{c(M^*(G))} = \frac{2\ell + \ell^* + 2(r + r^*) + 2r + r^*}{\ell^* + r^*} = \frac{2\ell + \ell^* + 4r + 3r^*}{\ell^* + r^*} \ .$$

Recalling that $D_L \geq D_R$, we express D_L as $D_L = (1 + \lambda)D_R$ for some $\lambda \geq 0$, and so $\ell = (1 + \lambda)r$ and $\ell^* = (1 + \lambda)r^*$. Thus,

$$\rho(\xi) = \frac{2(1 + \lambda)r + (1 + \lambda)r^* + 4r + 3r^*}{(1 + \lambda)r^* + r^*} = \frac{(6 + 2\lambda)r + (4 + \lambda)r^*}{(2 + \lambda)r^*} \ .$$

Assuming that the edge costs in G (as a whole) are scaled so that $R = H^k$ (rather than merely being consistent with H^k), and recalling the properties of ξ^k_{RT}, we get

$$\rho(\xi) = \frac{(6 + 2\lambda)(3^{k-1} - 2^{k-1}) + (4 + \lambda)2^{k-1}}{(2 + \lambda)2^{k-1}} = \left(\frac{6 + 2\lambda}{2 + \lambda}\right) \cdot (3/2)^{k-1} - 1 \ .$$

The lemma follows since the function $f(\lambda) = \frac{6+2\lambda}{2+\lambda}$ is monotonically decreasing for $\lambda \geq 0$, meaning that it assumes its maximum for $\lambda = 0$ which implies that $D_L = D_R$ has to hold. □

2.4 Lower Bound on PoS_α

Our goal in this section is to prove Theorem 2.2 and thereby establish a lower bound on PoS_α of minimum-cost perfect matching in metric graphs with $2n$ vertices.

Theorem 2.2. *PoS_α of minimum-cost perfect matching in metric graphs with $2n$ vertices is $\Omega(n^{\log(1+1/(2\alpha))})$.*

The graph construction that lies at the heart of this lower bound, denoted H_α^k, is a parametrized variant of the Reingold-Tarjan graph H^k presented in Section 2.3 and depicted in Figure 2.3 for arbitrary values of α. Specifically, the 2-vertex graph H_α^1 is identical to H^1; and the 2^{k+1}-vertex graph H_α^{k+1} is constructed recursively by placing two disjoint instances of H_α^k, each of diameter D_α^k, on the real line, only that this time, the spacing between them is set to $S_\alpha^{k+1} = (1/\alpha - \varepsilon)D_\alpha^k$, for some sufficiently small $\varepsilon > 0$ that will be determined later on. This implies that $D_\alpha^k = (2 + 1/\alpha - \varepsilon)^{k-1}$ and $S_\alpha^{k+1} = (1/\alpha - \varepsilon)(2 + 1/\alpha - \varepsilon)^{k-1}$.

Now let M be an α-stable matching in H_α^k. We argue that M has to contain each edge $e = (x, y)$ with $c(e) = 1/\alpha - \varepsilon$. Indeed, if $e \notin M$, then e is α-unstable in M since $c(e) < \alpha \cdot \min\{c(x, x'), c(y, y')\}$ for all other vertices x', y'. Given that all vertices with distance $1/\alpha - \varepsilon$ are therefore already matched, we can apply the same argument for each edge connecting two adjacent vertices with edge cost $(1/\alpha-\varepsilon)(2+1/\alpha-\varepsilon)$ and thereby conclude that these edges have to be in M as well. By repeating this argument, we end up with the unique α-stable matching M that has to contain the edge $(x_1^k, x_{2^k}^k)$ whose cost is D_α^k and and all other edges whose cost differs from 1. Thus, $c(M) \geq D_\alpha^k = (2 + 1/\alpha - \varepsilon)^{k-1}$.

On the other hand, the cost of the minimum-cost matching M^* is not larger than that of the matching using all cost 1 edges, thus we can bound the cost of M^* as $c(M^*) \leq 2^{k-1}$. Together, we conclude that

$$\begin{aligned} \mathrm{PoS}_\alpha(H_\alpha^k) \;&\geq\; \frac{c(M)}{c(M^*)} \;\geq\; \frac{(2 + 1/\alpha - \varepsilon)^{k-1}}{2^{k-1}} \\ &= \Omega\left(1 + \frac{1}{2\alpha}\right)^{k-1} \;=\; \Omega\left(n^{\log\left(1+\frac{1}{2\alpha}\right)}\right) \;, \end{aligned}$$

where the last two equalities hold by taking a sufficiently small ε and by recalling that H_α^k has $2n = 2^k$ vertices, resp.

2.5 Price of Stability

The upper bound established on the PoA in Section 2.3 clearly holds for the PoS, too. In Section 2.4, we showed that the proof technique for the $\Omega(n^{\log(3/2)})$-lower bound of Section 2.3 can be easily adapted to establish the same lower bound for the PoS as well, so the PoS does not provide much of an improvement over the PoA. In fact, Section 2.4 generalizes this result, showing that $\mathrm{PoS}_\alpha(2n) = \Omega(n^{\log(1+1/(2\alpha))})$ for every $\alpha \geq 1$. Consequently, we turn our attention to bounding $\mathrm{PoS}_\alpha(2n)$ from above, establishing the following theorem.

Theorem 2.3. *PoS_α of minimum-cost perfect matching in metric graphs with $2n$ vertices is at most $3 \cdot n^{\log(1+1/(2\alpha))}$.*

Observe that Theorem 2.2 and 2.3 establish a tight bound of $\Theta(n^{\log(1+1/(2\alpha))})$ on $\mathrm{PoS}_\alpha(2n)$. The upper bound promised by Theorem 2.3 is constructive, relying on a simple algorithm presented in Section 2.5.1. Section 2.5.2 provides the analysis of this algorithm, showing that the returned matching indeed satisfies the bound.

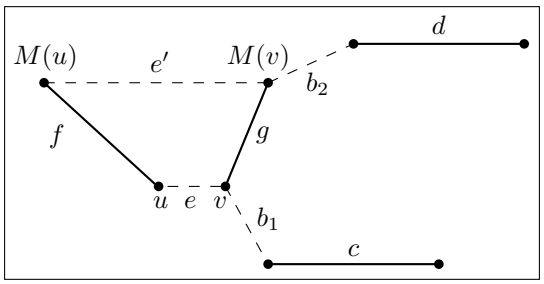


Figure 2.5: This figure illustrates the two different cases of Lemma 2.6.

2.5.1 An Algorithm for α-Stable Matchings

The following algorithm STAB transforms a minimum-cost matching M^* in a metric graph into an α-stable matching M.

ALGORITHM STAB: Start with the minimum-cost matching $M \leftarrow M^*$ and iterate over all edges of G by non-decreasing order of costs. If the edge (u, v) currently considered is α-unstable in the current matching M, replace the edges $(u, M(u))$ and $(v, M(v))$ in M by (u, v) and $(M(u), M(v))$ (this operation is called a *flip* of the edge (u, v)) and continue with the next edge. After having iterated over all edges, return M.

We assume that edge cost ties are resolved in an arbitrary but consistent manner. In the following, we denote by M_i the matching calculated by the above algorithm at the end of iteration i. Moreover, $M_0 = M^*$ is the initial minimum-cost matching and M_S the final matching returned by STAB.

Lemma 2.6. *For any unstable edge b created by the flip of an edge e, we have $c(b) > c(e)$.*

Proof. We consider a flip of the edge $e = (u, v)$ and denote by $e' = (M(u), M(v))$ the second new edge joining M as a result of the flip. The two edges that are removed by the flip are denoted by f and g. See Figure 2.5 for an illustration of the situation. When an edge e is flipped, there are essentially two different cases for an unstable edge to be created. The unstable edge contains either one vertex of e or one vertex of e'. No other vertices are involved in the flip and thus every *new* unstable edge has to contain at least one of the four vertices. We assume without loss of generality that a vertex of the edge g is incident to the unstable edge created by the flip.

Let us first consider the case where a vertex of e is incident to the new unstable edge. This case is denoted as the edge b_1 in Figure 2.5. We assume that b_1 is stable before the flip and unstable thereafter. For b_1 to be unstable after the flip, we must have $\alpha \cdot c(b_1) < c(e)$ and $\alpha \cdot c(b_1) < c(c)$. But as e is unstable before the flip, we have $\alpha \cdot c(e) < c(g)$ and thus we get $\alpha \cdot c(b_1) < c(e) < c(g)/\alpha \leq c(g)$. This means that b_1 was already unstable before the flip, which is a contradiction to the assumption. Hence, no vertex of e can be part of the new unstable edge.

Let us now consider the case, where a vertex from e' is part of the new unstable edge (b_2 in Figure 2.5). Since b_2 is stable before the flip and unstable

after it, we must have $c(g) \leq \alpha \cdot c(b_2) < c(e')$. But as e is unstable before the flip, we have $\alpha \cdot c(e) < c(g)$, and thus we get $c(e) < c(g)/\alpha \leq c(b_2)$ which completes the proof. □

Corollary 2.7 follows by induction on i. Lemma 2.8 then follows by a straightforward analysis of the algorithm's run-time.

Corollary 2.7. *Let e_i be the edge considered in iteration i. Then for any unstable edge b in M_i it holds that either $c(e_i) < c(b)$ or b will be considered in a later iteration $j > i$.*

Lemma 2.8. *Algorithm* Stab *transforms a minimum-cost matching into a valid α-stable matching in time $\mathcal{O}(n^2 \log n)$.*

Proof. The running time of the algorithm is dominated by the step of sorting the edges in G according to their cost. This takes $\mathcal{O}(n^2 \log n)$ steps. The second phase — the actual algorithm — runs in $\mathcal{O}(n^2)$ steps since it iterates once over all edges in $V \times V$ and each iteration takes $\mathcal{O}(1)$ time.

The correctness of the algorithm is established by Corollary 2.7 since it states that in the last iteration, all unstable edges have strictly larger cost than the edge currently considered or will be considered later. Since this edge is already the one with the largest cost and all edges have been considered, there cannot be any unstable edges in the final matching M_S. □

2.5.2 Cost Analysis

Our goal in this section is to show that when the algorithm Stab is invoked with parameter α for any $\alpha \geq 1$, it returns an α-stable matching M_S satisfying $c(M_S) = c(M^*) \cdot \mathcal{O}(n^{\log(1+1/(2\alpha))})$. Since this section makes heavy use of rooted binary trees and their properties, we require a few definitions. In a *full binary tree*, each inner node has exactly two children. The *depth* $d(v)$ of a node v in a tree T is the length of the unique path from the root of T to v and the *height* $h(T)$ of a tree T is defined as the maximal depth of any node in T. The *height* $h(v)$ of a node v of T is defined to be the height of its subtree. The *leaf set* $\mathcal{L}(T)$ or $\mathcal{L}(F)$ of a tree T or a collection F of trees is the set of all leaves in T or F, resp. The *leaf set* $\mathcal{L}(v)$ of a node v in a tree is $\mathcal{L}(T_v)$ where T_v is the subtree rooted at v. Finally, two nodes with the same parent are called *sibling nodes*.

We begin with Lemma 2.9 stating an important property of the edges that are flipped by Stab.

Lemma 2.9. *If an edge e is flipped in iteration i, then $e \in M_j$ for all $j \geq i$ and, in particular, $e \in M_S$.*

Proof. Let us assume for the sake of contradiction that $e = (u, v)$ was flipped in iteration i of the algorithm and further that $(u, v) \notin M_j$ for some $j > i$. According to the algorithm, we have $(u, v) \in M_i$. Since $(u, v) \notin M_j$, there has to exist an iteration k with $i < k \leq j$ where (u, v) is removed from M_{k-1} such that $(u, v) \notin M_k$. For this to happen, either edge (u, u') or (v, v') for some vertex u' or v' must be flipped in iteration k because it was unstable in M_{k-1}. Without loss

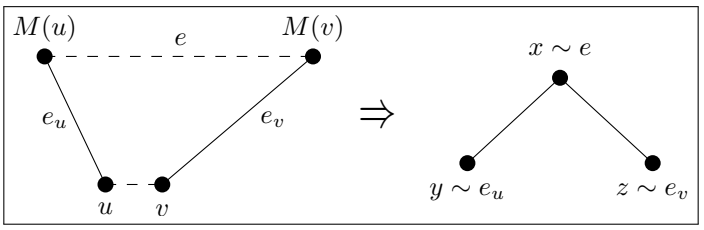


Figure 2.6: The left side shows a matching configuration with an unstable edge (u, v), which will be flipped by Stab. This flip is then represented by the flip tree segment on the right, which depicts the replacement of the two active edges $(u, M(u)) \sim y$ and $(v, M(v)) \sim z$ by the active edge $(M(u), M(v)) \sim x$.

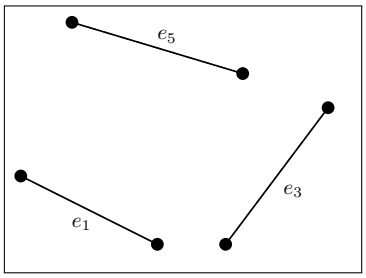
of generality, we assume that (u, u') is unstable in M_{k-1} and flipped in iteration $k > i$. Thus, we have $c(u, u') \leq \alpha \cdot c(u, u') < c(u, v)$. But this means that Stab would have considered the edge (u, u') before considering the edge (u, v), a contradiction to the assumption. □

Consider an iteration of Stab where edge (u, v) is flipped because it was unstable at the beginning of the iteration. Then the two edges $(u, M(u))$ and $(v, M(v))$ are replaced by (u, v) and $(M(u), M(v))$. Since the edge (u, v) is selected irrevocably according to Lemma 2.9, the edges $(u, M(u))$ and $(v, M(v))$ can never be part of M again. The only edge, of the four edges involved, that may be changed again, is the edge $(M(u), M(v))$. Thus, we refer to $(M(u), M(v))$ as an *active* edge. We also refer to all edges in M_0 as active. Using the notion of active edges, we shall now model the changes that Stab applies to the matching during its execution through a logical helper structure called the flip forest.

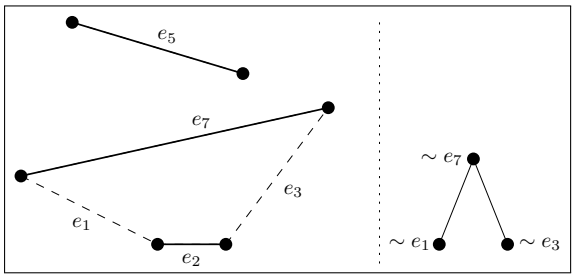
To avoid confusion between the basic elements of our graphs and the basic elements of the flip forest, we refer to the former as vertices/edges and to the latter as nodes/links.

Definition (Flip Forest)**.** The *flip forest* $F = (U, K)$ for a certain execution of Stab is a collection of disjoint rooted trees and has node set U and link set K. For each edge $e \in V \times V$ that has been active at some stage during the execution, there exists a node $u_e \in U$. This correspondence is denoted by $u_e \sim e$. For each flip of an edge (u, v) in G, resulting in the removal of the edges $(u, M(u))$ and $(v, M(v))$ from M, K contains a link connecting the node $y \sim (u, M(u))$ to its parent $x \sim (M(u), M(v))$ and a link connecting the node $z \sim (v, M(v))$ to its parent $x \sim (M(u), M(v))$. (Observe that, by definition, all three edges $(u, M(u))$, $(v, M(v))$, and $(M(u), M(v))$ are active.) Refer to Figure 2.6 for an illustration.

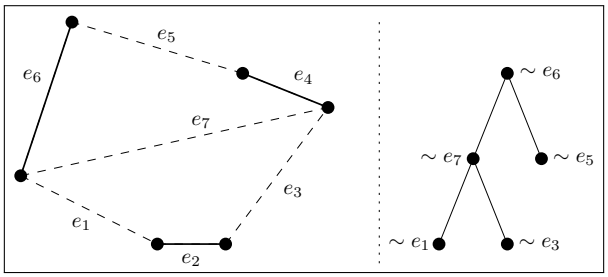
The definition of a flip forest ensures that for each flip of the algorithm, we obtain a binary *flip tree segment* as depicted by Figure 2.6. When we transcribe each flip operation of the complete execution of Stab into a flip tree segment as explained above, we end up with a collection of full binary trees — the *flip forest*. This is because the parent node of a tree segment may appear as a child node of the tree segment corresponding to a later iteration of the algorithm since its corresponding edge is still active and therefore may participate in another flip.



(a) The initial matching M_0 is a minimum-cost matching.



(b) Matching M_1 is obtained from M_0 by flipping e_2. The corresponding flip tree is shown on the right.



(c) Matching M_2 is obtained from M_1 by flipping e_4. The corresponding flip tree is shown on the right.

Figure 2.7: This figure shows how the initial minimum-cost matching M_0 is transformed by an execution of Stab through the flips of the edges e_2 and e_4 along with the flip forest (here only a single flip tree) corresponding to the execution. Edges in the current matching are drawn with solid lines while edges in matchings of previous iterations are drawn with dashed lines.

Each such tree is called a *flip tree* hereafter. Figure 2.7 illustrates a sample execution of Stab.

Observe that all leaves (including isolated nodes) in the flip forest correspond to edges in the minimum-cost matching $M_0 = M^*$. The edges in the matching M_S are implicitly represented by the flip forest: An edge that gets flipped — and is therefore irrevocably selected into M_S — has no corresponding node in F, but we may associate it with the node corresponding to the active edge resulting from the flip. On top of these edges, M_S contains the edges corresponding to isolated nodes or roots.

We now define a function $\psi : U \mapsto \mathbb{R}$ that assigns a real *weight* to each node in the flip forest F as follows. For each leaf ℓ of a flip tree in F, we set $\psi(\ell) \coloneqq c(e)$, where $\ell \sim e$ and we recall that an edge corresponding to a leaf node in F is part of M^*. The function ψ is extended to an inner node x of a flip tree with child nodes y and z by the recursion

$$\psi(x) \coloneqq \psi(y) + \psi(z) + (1/\alpha) \cdot \min\{\psi(y), \psi(z)\} \ . \tag{2.1}$$

For ease of notation, we call the child with smaller (resp., larger) weight as well as the link leading to its parent *light* (resp., *heavy*); ties are resolved arbitrarily. We denote the light child of a node x as x_L and the heavy child as x_H. Then we

can rewrite Equation (2.1) as

$$\psi(x) \coloneqq \psi(x_H) + (1 + 1/\alpha) \cdot \psi(x_L) \ .$$

Lemma 2.10. *Let x be a node in F and e an edge in G with $x \sim e$. Then $c(e) \leq \psi(x)$.*

Proof. We prove the statement by induction over the height of x in its flip tree. The assertion holds for every leaf $x \sim e$ in the flip forest as $\psi(x) = c(e)$ by definition. Assume that the statement holds for the two children x_L and x_H of a node x that represents a flip of the edge (u, v). Then $x \sim (M(u), M(v)) = e$ and we assume without loss of generality that $x_H \sim (u, M(u)) = e_u$ and $x_L \sim (v, M(v)) = e_v$. Thus, by the inductive hypothesis, $c(e_u) \leq \psi(x_H)$ and $c(e_v) \leq \psi(x_L)$. This flip tree segment represents the replacement of the edges e_u and e_v by e and (u, v), which happened because the edge (u, v) was unstable with respect to M, that is, $\alpha \cdot c(u, v) < \min\{c(e_v), c(e_u)\}$. Since G is metric, we can bound $c(e)$ as follows.

$$\begin{aligned} c(e) &\leq c(e_u) + c(e_v) + c(u, v) \\ &< c(e_u) + (1 + 1/\alpha) \cdot c(e_v) \\ &\leq \psi(x_H) + (1 + 1/\alpha) \cdot \psi(x_L) \qquad \text{(inductive hypothesis)} \\ &= \psi(x) \end{aligned}$$

□

At this stage, we would like to relate the weight $\psi(r_T)$ of the roots r_T in F to the cost of the stable matching M_S returned by Stab. To that end, we observe that M_S consists of the edges corresponding to the roots in F and to the edges that have been flipped along the course of the execution; let R and D denote the set of the former and latter edges, respectively. Observe that

$$c(M_S) = \sum_{e \in R} c(e) + \sum_{e \in D} c(e) \ .$$

Consider the flip of the edge (u, v) resulting in the insertion of the edge $(M(u), M(v)) \sim x$ to M and the removal of the edges $(u, M(u)) \sim x_L$ and $(v, M(v)) \sim x_H$ from M. Since $\psi(x) = \psi(x_H) + (1 + 1/\alpha) \cdot \psi(x_L)$, we have $\psi(x) - (\psi(x_L) + \psi(x_H)) = \psi(x_L)/\alpha$. Lemma 2.10 then implies that $\psi(x) - (\psi(x_L) + \psi(x_H)) \geq c(u, M(u))/\alpha$, and since edge (u, v) was flipped, we have $\psi(x) - (\psi(x_L) + \psi(x_H)) \geq c(u, v)$. Therefore,

$$\begin{aligned} \sum_{e \in D} c(e) &\leq \sum_{\text{internal } x \in U} (\psi(x) - (\psi(x_L) + \psi(x_H))) \\ &= \sum_{\text{flip trees T}} \Big(\psi(r_T) \ - \sum_{\ell \in \mathcal{L}(T)} \psi(\ell) \Big) \\ &= \sum_{\text{flip trees T}} \psi(r_T) \ - \sum_{\ell \in \mathcal{L}(F)} \psi(\ell) \ , \end{aligned}$$

where the second equation holds by a telescoping argument. Note further that $\sum_{e \in R} c(e) \leq \sum_{\text{flip trees} T} \psi(r_T)$ and thus

$$c(M_S) \leq 2 \sum_{\text{flip trees T}} \psi(r_T) \ - \sum_{\ell \in \mathcal{L}(F)} \psi(\ell) \ .$$

Since $c(M^*) = \sum_{\ell \in \mathcal{L}(F)} \psi(\ell)$, Corollary 2.11 follows.

Corollary 2.11. *The matching M_S returned by* Stab *satisfies*

$$c(M_S) \leq 2 \sum_{\text{flip trees } T} \psi(r_T) - c(M^*) \ .$$

We will now have a closer look at the properties of our flip trees and their weights. It will be convenient to ignore the relation of the flip trees to the Stab algorithm at this stage; in other words, we consider an abstract full binary tree T with a function $w : \mathcal{L}(T) \to \mathbb{R}_0^+$ that assigns non-negative real weights to the leaves of T. For any leaf ℓ of T, we set $\psi(\ell) = w(\ell)$ and determine the weight $\psi(x)$ of each inner node x in T following the recursion given by Equation (2.1). Note that we allow our tree T to have zero-weight leaves now (this can only make our analysis more general).

Definition (Complete Binary Tree)**.** A full binary tree T is called *complete* if all leaves are at depth $h(T)$ or $h(T) - 1$. Given some positive integer n that will typically be the number of leaves in some tree, let

$$h(n) = \lceil \log n \rceil \quad \text{and} \quad k(n) = 2^{h(n)} - n \ .$$

Note that $0 \leq k(n) < 2^{h(n)-1}$.

Observe that for a complete full binary T with n leaves, $h(n)$ equals the height $h(T)$ of T while $k(n)$ equals the number of missing leaves at the maximum depth $h(T)$.

Definition (ψ-Balanced Binary Tree)**.** A full binary tree T is called *ψ-balanced* if for any two sibling nodes x, y in T, we have $\psi(x) = \psi(y)$.

Consider a full binary tree T. Let $\Lambda(T)$ denote the sum of the weights of the leaves of T, i.e., $\Lambda(T) = \sum_{\ell \in \mathcal{L}(T)} \psi(\ell)$, and let $\Psi(T) = \psi(r_T)$ (recall that r_T denotes the root of T). The following observation is established by induction on the node depth.

Observation 2.12. *For any node v of a ψ-balanced full binary tree T, we have*

$$\psi(v) = (2 + 1/\alpha)^{-d(v)} \cdot \Psi(T) \ .$$

Definition (Effect of Binary Tree)**.** The *effect* $\eta(T)$ of a full binary tree T is defined to be

$$\eta(T) \ = \ \begin{cases} \Psi(T)/\Lambda(T) & \text{if } \Lambda(T) > 0 \\ 1 & \text{if } \Lambda(T) = 0 \end{cases} \ .$$

An n-leaf full binary tree T is said to be *effective* if it maximizes $\eta(T)$, i.e., if there does not exist any n-leaf full binary tree T' such that $\eta(T') > \eta(T)$.

Intuitively speaking, if we think of T as a flip tree, then its effect is a measure for the factor by which the flips represented by T increase the cost of M^* when applied to it. But, once again, we do not restrict our attention to flip trees at this

stage. The effect of a full binary tree is essentially determined by its topology and by the assignment of weights to its leaves.

Before we proceed towards proving the upper bound on $\mathrm{PoS}_\alpha(2n)$, we first establish that the effect of a flip tree is invariant under scaling the leaf weights. We begin by defining the *light depth* $\lambda(x)$ of a node x in a flip tree T as the number of light links on the direct path from x to the root r_T of T. Lemma 2.13 relates the weight of a node of a flip tree to the weights of the leaves of its subtree.

Lemma 2.13. *Every node x in a flip tree satisfies*

$$\psi(x) \;=\; \sum_{\ell \in \mathcal{L}(x)} (1+1/\alpha)^{\lambda(\ell)-\lambda(x)} \cdot \psi(\ell) \;\;.$$

Proof. We prove the statement by induction over the height of x in its flip tree. The statement holds for a leaf node x since then we have $\mathcal{L}(x) = \{x\}$ and $\lambda(x) - \lambda(x) = 0$. Assume that the statement holds for both children $x_{\scriptscriptstyle H}$ and $x_{\scriptscriptstyle L}$ of a node x. By definition, we have

$$\begin{aligned}
\psi(x) &= \psi(x_{\scriptscriptstyle H}) + (1+1/\alpha)\cdot\psi(x_{\scriptscriptstyle L}) \\
&= \sum_{\ell \in \mathcal{L}(x_{\scriptscriptstyle H})} (1+1/\alpha)^{\lambda(\ell)-\lambda(x_{\scriptscriptstyle H})} \cdot \psi(\ell) \\
&\quad + (1+1/\alpha)\cdot \sum_{\ell \in \mathcal{L}(x_{\scriptscriptstyle L})} (1+1/\alpha)^{\lambda(\ell)-\lambda(x_{\scriptscriptstyle L})} \cdot \psi(\ell) \\
&= \sum_{\ell \in \mathcal{L}(x_{\scriptscriptstyle H})} (1+1/\alpha)^{\lambda(\ell)-\lambda(x)} \cdot \psi(\ell) \\
&\quad + (1+1/\alpha)\cdot \sum_{\ell \in \mathcal{L}(x_{\scriptscriptstyle L})} (1+1/\alpha)^{\lambda(\ell)-\lambda(x)-1} \cdot \psi(\ell) \\
&= \sum_{\ell \in \mathcal{L}(x_{\scriptscriptstyle H})} (1+1/\alpha)^{\lambda(\ell)-\lambda(x)} \cdot \psi(\ell) + \sum_{\ell \in \mathcal{L}(x_{\scriptscriptstyle L})} (1+1/\alpha)^{\lambda(\ell)-\lambda(x)} \cdot \psi(\ell) \\
&= \sum_{\ell \in \mathcal{L}(x)} (1+1/\alpha)^{\lambda(\ell)-\lambda(x)} \cdot \psi(\ell) \;\;,
\end{aligned}$$

where we used $\lambda(x_{\scriptscriptstyle L}) = \lambda(x) + 1$ and $\lambda(x_{\scriptscriptstyle H}) = \lambda(x)$. □

Lemma 2.14. *The effect of a flip tree is invariant under scaling its leave weights.*

Proof. Recall that, by definition, $\Psi(T) = \psi(r_T)$ where r_T is the root of T and $\Lambda(T) = \sum_{\ell \in \mathcal{L}(r_T)} \psi(\ell)$. Lemma 2.13 then yields $\Psi(T) = \sum_{\ell \in \mathcal{L}(r_T)} (1+1/\alpha)^{\lambda(\ell)} \cdot \psi(\ell)$ since $\lambda(r_T) = 0$ for the root node. The effect of a flip tree T can then be expressed as

$$\eta(T) = \frac{\Psi(T)}{\Lambda(T)} = \frac{\sum_{\ell \in \mathcal{L}(r_T)} (1+1/\alpha)^{\lambda(\ell)} \cdot \psi(\ell)}{\sum_{\ell \in \mathcal{L}(r_T)} \psi(\ell)}$$

and the claim follows. □

Our upper bound is established by showing that the effect of an effective n-leaf full binary tree is $\mathcal{O}\big(n^{\log(1+1/(2\alpha))}\big)$. We begin by developing a better understanding of the topology of effective ψ-balanced full binary trees.

Lemma 2.15. *An effective n-leaf ψ-balanced full binary tree must be complete.*

Proof. Aiming for a contradiction, suppose that T is not complete. We scale the leaf weights of T so that $\Psi(T) = 1$, knowing by Lemma 2.14 that this does not change the effect of T. Because T is not complete, it must have leaves at depth d_1 and at depth d_2, where $d_2 > d_1 + 1$. The assertion is established by showing that an n-leaf full binary tree with higher effect can be obtained by a small modification to T's topology, in contradiction to the assumption that T is effective.

Let y be a leaf at depth d_1 and ℓ_1 and ℓ_2 be two leaves at depth $d_2 > d_1 + 1$ with parent node z. Since T is ψ-balanced, we can employ Observation 2.12 to conclude that

$$\psi(\ell_1) = \psi(\ell_2) = (2 + 1/\alpha)^{-d_2} \quad \text{and} \quad \psi(y) = (2 + 1/\alpha)^{-d_1} \ .$$

Now, consider the ψ-balanced full binary tree T' obtained from T by removing ℓ_1 and ℓ_2 and adding two new leaves ℓ'_1 and ℓ'_2 as children of y with weights $\psi(\ell'_1) = \psi(\ell'_2) = (2 + 1/\alpha)^{-d_1-1}$, keeping the weights of all other nodes unchanged. By doing so, we turn z — an internal node in T — into a leaf (whose weight remains $\psi(z) = (2+1/\alpha)^{-d_2+1}$). On the other hand, y, which is a leaf in T, is an internal node in T'. Therefore,

$$\begin{aligned}
\Lambda(T') &= \Lambda(T) + \psi(\ell'_1) + \psi(\ell'_2) + \psi(z) - \psi(\ell_1) - \psi(\ell_2) - \psi(y) \\
&= \Lambda(T) + 2 \cdot (2 + 1/\alpha)^{-d_1-1} + (2 + 1/\alpha)^{-d_2+1} \\
&\qquad - 2 \cdot (2 + 1/\alpha)^{-d_2} - (2 + 1/\alpha)^{-d_1} \\
&= \Lambda(T) + (2 + 1/\alpha)^{-d_1-1}(2 - (2 + 1/\alpha)) \\
&\qquad + (2 + 1/\alpha)^{-d_2}(2 + 1/\alpha - 2) \\
&= \Lambda(T) + (1/\alpha)((2 + 1/\alpha)^{-d_2} - (2 + 1/\alpha)^{-(d_1+1)}) \\
&< \Lambda(T) \ .
\end{aligned}$$

As $\Psi(T') = \Psi(T) = 1$, it follows that $\eta(T') > \eta(T)$, in contradiction to the effectiveness of T. □

Next, we develop a closed-form expression for the effect of complete ψ-balanced full binary trees. We define the function $\varphi : \mathbb{Z}^+ \mapsto \mathbb{R}$ with

$$\varphi(n) := \frac{(2 + 1/\alpha)^{h(n)}}{2^{h(n)} + k(n)/\alpha} \ .$$

and recall that $h(n) = \lceil \log n \rceil$ and $k(n) = 2^{h(n)} - n$.

Lemma 2.16. *The effect of an n-leaf complete ψ-balanced full binary tree T is*

$$\eta(T) = \varphi(n) \ .$$

Proof. Again we assume without loss of generality that the weights of the leaves are scaled so that $\Psi(T) = 1$, cf. Lemma 2.14. By definition, T has $2^h - 2k$ leaves at depth h and k leaves at depth $h-1$. Employing Observation 2.12, we conclude

$$\begin{aligned}\Lambda(T) &= (2^h - 2k) \cdot (2 + 1/\alpha)^{-h} + k \cdot (2 + 1/\alpha)^{-(h-1)} \\ &= (2 + 1/\alpha)^{-h} \cdot (2^h - 2k + k \cdot (2 + 1/\alpha)) \\ &= (2 + 1/\alpha)^{-h} \cdot (2^h + k/\alpha) \ .\end{aligned}$$

Since $\Psi(T) = 1$, we have $\eta(T) = 1/\Lambda(T)$ which completes the proof. □

An important property of complete ψ-balanced full binary trees is that their effect is strictly increasing in the number of leaves.

Lemma 2.17. *The function $\varphi(n)$ is strictly increasing.*

Proof. We show that for all $n \in \mathbb{Z}^+$, it holds that $\varphi(n+1) > \varphi(n)$ and distinguish two cases. First, we consider the case that $n \neq 2^i$ for all $i \in \mathbb{Z}^+$. Observe that $h(n+1) = h(n)$ and $k(n+1) < k(n)$ and therefore $\varphi(n+1) > \varphi(n)$. Now, we examine the case that $n = 2^i$ for some $i \in \mathbb{Z}^+$. We have $h(n+1) = i + 1$ and $h(n) = i$ as well as $k(n+1) = 2^i - 1$ and $k(n) = 0$. Plugging these values into φ, we obtain $\varphi(n+1) > \varphi(n)$ and the proof is complete. □

We are now ready to show that it is sufficient to consider complete ψ-balanced full binary trees.

Lemma 2.18. *An effective n-leaf full binary tree must be ψ-balanced.*

Proof. We prove the statement by induction on the number of leaves n. The base case of a tree having a single leaf (which is also the root) holds vacuously; the base case of a tree having two leaves is trivial. Assume that the assertion holds for trees with fewer than n leaves and let T be an effective n-leaf full binary tree. Let T_ℓ and T_r be the left and right subtrees of T and let z be the number of leaves in T_ℓ where $1 \leq z \leq n-1$. Observe that both T_ℓ and T_r have to be effective as otherwise, $\eta(T)$ could be increased; more precisely, if $T_i \in \{T_\ell, T_r\}$ is not effective, then one can increase $\Psi(T_i)$ while keeping $\Lambda(T_i)$ unchanged, which results in an increased $\eta(T)$. Thus, by the inductive hypothesis, we can assume that T_ℓ and T_r are ψ-balanced. Lemma 2.15 then guarantees that both T_ℓ and T_r are complete. Hence, we can use Lemma 2.16 to determine the effects of T_ℓ and T_r as $\varphi(z)$ and $\varphi(n-z)$, respectively.

Assume without loss of generality that the leaf weights are scaled such that $\Lambda(T) = \Lambda(T_\ell) + \Lambda(T_r) = 1$ and set $\Lambda(T_\ell) = x$, $\Lambda(T_r) = 1 - x$, for some $0 \leq x \leq 1$. We consider a set of $n-1$ function $f_z : [0,1] \mapsto \mathbb{R}_{>0}$ (parametrized by z) with

$$f_z(x) = \begin{cases} \varphi(z) \cdot x + (1 + 1/\alpha)\varphi(n-z) \cdot (1-x) & \text{if } \varphi(z)x \geq \varphi(n-z)(1-x) \\ (1 + 1/\alpha)\varphi(z) \cdot x + \varphi(n-z) \cdot (1-x) & \text{if } \varphi(z)x \leq \varphi(n-z)(1-x) \end{cases}$$

that, by Lemma 2.16, determines the effect of T given x and z. Observe that each f_z is a piecewise linear and continuous function, which is linear in the

intervals $[0, b_z]$ and $[b_z, 1]$ where b_z is the break point of f_z such that $\varphi(z)b_z = \varphi(n-z)(1-b_z)$. Hence, f_z must attain its maximum either at a boundary point 0 or 1, or at the breakpoint b_z, where the latter is realized by a ψ-balanced tree.

Consider the function $f(x) = \max_z f_z(x)$ whose maximum corresponds to the effect of an effective n-leaf full binary tree. Let $\hat{x}$ be the argument for which $f(x)$ is maximized and we argue that $\hat{x}$ can be neither 0 nor 1. Indeed, if $\hat{x} = 0$, then $\Psi(T) = \Psi(T_r)$ and $\Lambda(T) = \Lambda(T_r)$, hence $\eta(T) = \eta(T_r)$ for the corresponding tree T. But since T_r has fewer leaves than T and is complete and ψ-balanced, Lemma 2.16 and 2.17 dictate that its effect — and thus also the effect of T — must be smaller than the effect of an n-leaf complete ψ-balanced full binary tree, a contradiction to the choice of $\hat{x}$ maximizing $f(x)$. An analogous argument excludes $\hat{x} = 1$. It follows that the maximum of $f(x)$ must be attained at a point $0 < \hat{x} < 1$, which, by the definition of f, is the break point b_z of some function f_z and thus realized by a ψ-balanced tree. □

Combining Lemmas 2.15, 2.16, and 2.18 and recalling that $h = h(n) = \lceil \log n \rceil \leq \log n + 1$ and $k = k(n) \geq 0$, we conclude that the effect of an n-leaf full binary tree is at most

$$\frac{(2+1/\alpha)^h}{2^h + k/\alpha} \;\leq\; \frac{(2+1/\alpha)^h}{2^h} \;\leq\; (1+1/(2\alpha))^{\log n+1} \;\leq\; 3/2 \cdot n^{\log(1+1/(2\alpha))} \;.$$

Returning to the definition of the flip forest F, we recall that there exists one leaf in F for each of the n edges in the minimum-cost matching M^* and therefore each flip tree has at most n leaves. Furthermore, since

$$c(M^*) = \sum_{\text{flip trees } T} \sum_{\ell \in \mathcal{L}(T)} \psi(\ell) = \sum_{\text{flip trees } T} \Lambda(T) \;,$$

we can employ Corollary 2.11 to derive

$$\frac{c(M_S)}{c(M^*)} \;\leq\; 2 \cdot \frac{\sum_{\text{flip trees } T} \Psi(T)}{\sum_{\text{flip trees } T} \Lambda(T)} \;\leq\; 2 \cdot \max_{\text{flip trees } T} \eta(T) \;\leq\; 3 \cdot n^{\log(1+1/(2\alpha))} \,,$$

thus establishing Theorem 2.3.

2.6 Conclusion

"In matching, stability is not for free" puts the results of this chapter in a nutshell. We have shown that in order to discourage nodes from deviating from a suggested assignment, one might have to pay a high price in terms of the overall cost of all matched pairs. However, when looking at real-world instances of matchings such as relationships between humans, it becomes apparent pretty quickly that 1-stability is usually not the most appropriate solution concept. Who would really be willing to leave one's partner for someone who is just infinitesimally "better"? When looking at α-stability, our results are much more encouraging as the overall costs only increase by a constant factor if one is

satisfied with a $\mathcal{O}(\log n)$-stable matching. Contemplated the other way around, paying only a constant factor already yields a significant level of stability.

We hope that our findings can assist real-word matching systems such as matching donated organs to organ receivers, doctors to hospitals, people in dating platforms to each other while ensuring in particular that entities are happier with their matches and have less incentives to deviate, whatever that means in the respective context.

Part II

Mobile Agents with Restricted Capabilities

3

Gathering of Mobile Robots with Limited Visibility

In the future, large groups of small and cheap mobile robots can potentially replace few and expensive robots for many tasks. Thus, there is a growing interest in figuring out which kinds of tasks can be solved by such robotic teams. For mobile robots, it is especially interesting whether they can build a given formation and which capabilities are needed to do so. Naturally, the goal is to require as few capabilities as possible in order to be able to use robots that are as cheap as possible.

In this chapter we study the robot-gathering problem, a classic mobile network problem. As we discuss in more detail in the related work section, robot-gathering has received considerable attention in the past few years, and there exist various model variants. We are particularly interested in the concurrent version of the problem: We are given n robots, modeled as points in the two-dimensional Euclidean plane, and these robots want to gather at a single point. In each synchronous round, every robot observes the plane and the other robots, decides where to move, and moves there, concurrently with all other robots. The robots are oblivious and thus cannot remember any information between rounds. The next round does not start before the last movement has finished. If robots have full visibility, the problem is trivial as all robots can compute the unique center of the smallest enclosing circle (SEC) of all robots, and then concurrently move there, finishing in one single round. Hence, we study the distributed version of the problem where each robot has a limited viewing range and can only observe other robots that are within unit distance of its position. This notion implies that the visibility graph of the robots is a unit disk graph (UDG). Clearly, the UDG of the robots must be connected initially, meaning that there is a path

from any robot to any other robot just following the visibility neighborhoods. Additionally we assume that robots are anonymous, in the sense that they do not have unique IDs. Again, if robots have unique IDs, the problem becomes much simpler, as the robots just have to agree on meeting at the location of the robot with the minimum ID.

The most important question in the aforementioned model is whether the robots are able to meet at a single point and how long it takes to do so. The answer to the first question is known for 15 years. In their seminal paper Ando, Suzuki, and Yamashita [10] presented an algorithm that gathers the robots. In each round, every robot simply moves to the center of the SEC of the robots in its viewing range, only constrained by the condition that robots must not lose visibility to their neighboring robots. As Ando et al. proved, this approach works, and the robots eventually meet.

More recently, Chazelle [27] showed that similar processes may have an exponential behavior. It is therefore an interesting task to examine runtime bounds of the original SEC algorithm by Ando et al. In this chapter we show that the algorithm gathers all robots at a single point in a number of rounds polynomial in the number of nodes n, in particular $\mathcal{O}(n^2)$. Furthermore, we give a matching lower bound of $\Omega(n^2)$ and thus present a tight analysis of the SEC algorithm, showing that the algorithm needs $\Theta(n^2)$ rounds to gather all robots.

3.1 Related Work

The problem of gathering a set of robots has gained a lot of interest during the last 15 years. In early work, all robots had a global view of the positions of the other robots [106, 107]. Several articles have been published for the fully asynchronous and continuous setting, where the robots do not have a common notion of time, and hence may also observe each other while moving. A promising approach seems to let all robots move to the *Weber* point that — unlike the center of gravity or the center of the SEC — is invariant to movements of robots towards it. However, Bajaj [23] showed that the Weber point cannot be computed because it involves calculating roots of high-order polynomials. Cieliebak et al. [33] gave an algorithm that solves the gathering problem if the robots are able to detect whether there is more than one robot at a given point (multiplicity detection). Cohen et al. showed that moving to the center of gravity of the robots leads to convergence, even in highly asynchronous models [34,35]. Furthermore, Izumi et al. [66] showed exponential lower bounds for the convergence of a certain class of randomized algorithms.

We are mainly interested in the local model with limited visibility, where the robots have to base their decisions only on the positions of the neighboring robots within a given range. This setting is more difficult, because a robot does not know the system as a whole, often not even the total number of robots. Furthermore, it is essential to always guarantee the connectivity of the neighborhood graph, given that it is connected in the beginning. Otherwise it cannot be ensured that the connectivity will ever be regained — at least when dealing with oblivious robots. This is especially an issue in a synchronous and discrete

round model, which is common in the literature [10, 45, 106] and which we also consider in this chapter. As the robots move at the same time (possibly based on different information), it is difficult to keep the connectivity.

The gathering problem in the local setting was already tackled some time ago by Ando et al. [10]. Similar to other local algorithms for the gathering problem, their robots move to the center of the smallest enclosing circle of their neighbors' locations. This target point definition guarantees that connectivity is maintained if no two robots are activated at the same time. But it can be easily seen that connectivity is not necessarily maintained in the synchronous setting. To overcome this problem, the authors restrict the distance that a robot moves towards its target point in a clever way, such that connectivity is guaranteed even under worst-case movement of the other robots performing the same algorithm. Furthermore, Ando et al. showed that their algorithm allows the robots to gather in a finite number of rounds. Beyond this result, no runtime bounds were given. A follow-up article [9] evaluated the quality of their algorithm in a more realistic environment, where sensor data is not perfectly accurate, and suggested that the algorithm is robust against measurement errors of the sensors.

The same algorithm, but in an asynchronous setting, is used by Meyer auf der Heide et al. [81]. Here, the robots only move one at a time, and so no connectivity maintenance is required. It is shown that the robots also gather in this setting, but again, no runtime bounds are given.

Flocchini et al. [57] showed that having a common orientation among the robots is sufficient to solve the gathering problem in finite time in the fully asynchronous model. The work by Degener et al. [40] is closest to our new contribution. It is shown that gathering can be achieved in expected $\mathcal{O}(n^3 \log n)$ rounds if the robots move sequentially: in each step only one robot (chosen uniformly at random) is activated. Moreover, when active, robots do not only move themselves, but they need the additional capability to assign new target points to neighbors, which may then move as well. This is a very powerful assumption, since it enables a robot to move several robots to the same position and let them act like one single robot from then on.

Apart from this result, there are no runtime bounds known for other algorithms for the local gathering problem so far.

Other researchers have analyzed how the algorithms can cope with failures or inaccuracies of sensor readings. Among others, Souissi et al. [104] and Izumi et al. [67] presented algorithms that are able to deal with erroneous readings from a compass. Agmon et al. [1] studied algorithms that tolerate the crash of a single robot, and still are able to achieve gathering of the remaining robots.

The more general problem of constructing geometrical formations with a set of autonomous robots has also attracted a lot of research. Current work shows how these robots can form lines between fixed stations [38, 45, 46, 72, 76] or circles [26, 37].

In this chapter, we provide a lower bound of $\Omega(n^2)$ and, as our main result, a matching upper bound of $\mathcal{O}(n^2)$ for the number of rounds required to gather the robots using the local algorithm for the synchronous setting presented by Ando et al. [10]. The robots used here are considerably weaker than those discussed in

the work of Degener et al. [40], as they they cannot instruct any robots to move and are not allowed to view any further than their communication range.

Note, that the capabilities we require are quite restrictive compared to related work from robot formation problems. Other capabilities that are considered are for instance compasses [67, 104] and other time models such as the semi-synchronous model, where arbitrary subgroups of robots move synchronously [42].

3.2 Model

Our model is essentially the one defined by Ando et al. [10]. Given a set $\mathcal{R}$ of n robots $r_1, \ldots, r_n$ in the Euclidean plane, the goal is to gather all robots in one point. A robot is represented as a singular point in the plane, which means that robots cannot block each other's views or paths. We use a discrete, synchronous time model: In each round t, $t \in \mathbb{Z}_0^+$, all robots act synchronously at the same time. We call the positions $p_1(t), \ldots, p_n(t)$ of the robots at the beginning of round t the *configuration* at time t. When the round t under consideration is clear from the context, we will sometimes identify a robot r_i with its position $p_i(t)$. We further call the configuration at time 0 the *start configuration*. When we say *time* t, we refer to the beginning of round t. The (Euclidean) distance between two robots r_i and r_j is indicated by $d(p_i(t), p_j(t))$ or also by $d(r_i, r_j)$ when t is clear from the context. Two robots r_i and r_j can see each other, if $d(r_i, r_j) \leq 1$, where we call r_i and r_j *neighbors* and the distance 1 the *viewing range* of the robots. The set of all neighbors of a robot r_i — its *neighborhood* — at time t is denoted as $N_t(r_i)$ or just $N(r_i)$ if the time is clear from the context. The notion of limited visibility induces a unit disk graph, the *visibility graph* $UDG_t = (\mathcal{R}, E_t)$, where $(r_i, r_j) \in E_t$ iff r_i and r_j are mutually visible at time t, i.e., $\text{dist}(r_i(t), r_j(t)) \leq 1$. We will furthermore use the convex hull of a set of robot positions to which we will also refer by the convex hull of these robots.

We measure the quality of the algorithm by counting the number of synchronous rounds until the robots have gathered in one point. During each round, the robots act according to the *Look-Compute-Move* (LCM) model: First all robots synchronously observe their environment and determine the positions of their neighbors relative to their own position (Look-operation). During the Compute-operation, they use the observed positions as input for the algorithm described in Section 3.3. The algorithm outputs the point to which the robots move concurrently during the following Move-operation.

The algorithm is based on the smallest enclosing circle (SEC) of a point set $\mathcal{P}$ (which are robot positions in our context). Its center is the point that minimizes the maximum distance to any point in $\mathcal{P}$.

Robot Model. Our robots have a limited viewing range, they are oblivious, which means that they do not have a memory, they do not communicate and they do not use a common coordinate system. Moreover, they cannot be distinguished from each other — they are anonymous. On the other hand, we abstract from technical issues. In particular, we assume the robots to be able to measure

positions of neighbors relative to their own position accurately, they can compute geometric properties and they can occupy the same position as other robots.

3.3 The Gathering Algorithm

1: // compute target point
2: $\mathcal{R}_i(t) :=$ {all robots visible from r_i including r_i itself}
3: $\mathcal{C}_i(t) :=$ smallest enclosing circle of $\mathcal{R}_i(t)$
4: $c_i(t) :=$ center of $\mathcal{C}_i(t)$
5: // keep connectivity
6: $\forall r_j \in \mathcal{R}_i(t) : m_j :=$ midpoint between $p_i(t)$ and $p_j(t)$
7: $\forall r_j \in \mathcal{R}_i(t) : \mathcal{D}_j :=$ circle with radius $\frac{1}{2}$ around m_j
8: seg := line segment $\overline{p_i(t), c_i(t)}$
9: $\mathcal{A} := \bigcap_{r_j \in \mathcal{R}} \mathcal{D}_j \cap$ seg
10: $x :=$ point in A that minimizes $d(x, c_i(t))$
11: // Note that $\mathcal{A} \neq \emptyset$, since $p_i(t) \in \mathcal{A}$
12: $p_i(t+1) := x$

Algorithm 3.1: Algorithm of robot r_i in round t

The algorithm introduced by Ando et al. [10], works as follows. First, r_i computes its *target point* $c_i(t)$, which is the center of the smallest enclosing circle around itself and its neighbors. Because the connectivity of the unit disk graph could break if all robots would move to their target point, a second phase is used to compute a point x on the line segment between $p_i(t)$ and $c_i(t)$ to which r_i finally moves. For each neighbor r_j, r_i computes the midpoint m_j between their positions and the *limit circle* D_j with center m_j and radius 1/2. As long as both r_i and r_j do not leave this circle, they will be in distance 1 of each other and therefore neighbors at the beginning of the next round. Finally, x is the point on the line segment between $p_i(t)$ and $c_i(t)$ that maximizes the distance that r_i moves under the constraint that r_i does not leave the circle D_j for any neighbor r_j. Since all robots execute this algorithm, this procedure makes sure that two neighboring robots never lose their connection.

Lemma 3.1 (Ando et al. [10])**.** *If two robots are neighbors in UDG_t at time t, then they are still neighbors in UDG_{t+1}. In particular, if UDG_0 is connected, then UDG_t is connected for all $t \geq 0$.*

Because of the procedure to keep connectivity, it is possible that a robot does not move far in direction towards its target point. We say that a robot r_j *hinders* another robot r_i from reaching some point p on the line segment between $p_i(t)$ and $c_i(t)$, if r_i would leave D_j when moving to p. If in any round, two robots move to the exact same point, they will stay at a common point for the rest of the execution of the algorithm, because they see the same neighborhood and hence behave exactly the same. We say that such robots have *merged*.

In [10], the authors have already shown that this algorithm gathers the robots in one point within finite time, but so far no runtime bounds were known. We

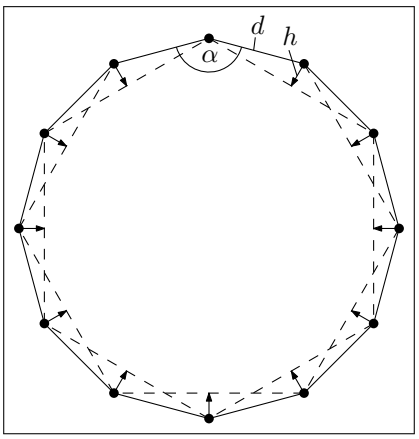


Figure 3.1: A robot configuration on the vertices of a regular convex polygon yields a worst-case running time of the algorithm.

will now first show a lower bound $\Omega(n^2)$, and then our main result, namely the upper runtime bound of $\mathcal{O}(n^2)$ rounds.

3.4 The Lower Bound

For a lower bound on the number of rounds until gathering when using the algorithm described in Section 3.3, consider a configuration with the robots positioned on the boundary of a circle, such that each robot has only two neighbors and the distance between two neighbors on the circle is the same for all robots. In this configuration, all robots have the same local view and so all robots do the same. The robots will therefore still be positioned on the boundary of a circle in the next round. We will use this observation to prove the following result.

Theorem 3.1. *There is a start configuration such that the algorithm takes $\Omega(n^2)$ rounds to gather the robots in one point.*

Proof. Let the robots be positioned on a circle with an initial distance of 1 between two neighboring robots (see Figure 3.1 for an illustration). This means that the initial circumference of the circle is $\approx n$, and its radius is $\approx \frac{n}{2\pi}$. We will show that it takes $\Omega(n^2)$ rounds until the circumference of the circle is reduced to $\frac{2}{3}n$.

If the circumference of the circle is greater than $\frac{2}{3}n$, each robot r has only two neighbors, which are in equal distance d, $\frac{1}{2} < d \leq 1$, from r. The center of the SEC of r's neighborhood is the midpoint between its neighbors. We can therefore compute the distance that r moves as the height h of the equilateral triangle formed by r and its two neighbors. To compute h, let α be the internal angle of the triangle at robot r. Due to the definition of the cosine, $h = \cos(\frac{\alpha}{2}) \cdot d$. In the interval between 0 and $\frac{\pi}{2}$, the cosine can be upper bounded by $\cos(x) \leq -x + \frac{\pi}{2}$. As $0 < \frac{\alpha}{2} < \frac{\pi}{2}$, we can apply this bound and thus $\cos(\frac{\alpha}{2}) \leq -\frac{\alpha}{2} + \frac{\pi}{2}$, resulting in $h \leq (-\frac{\alpha}{2} + \frac{\pi}{2}) \cdot d$. Moreover, since the robots form a regular polygon with n

vertices and the sum of the internal angles of such a polygon is $\pi n - 2\pi$, we get that $\alpha = \pi - \frac{2\pi}{n}$ for all robots. Thus,

$$\begin{aligned} h &\leq \left(-\frac{\alpha}{2} + \frac{\pi}{2}\right) \cdot d \\ &\leq \left(-\left(\frac{\pi}{2} - \frac{\pi}{n}\right) + \frac{\pi}{2}\right) \cdot d \\ &= \frac{\pi}{n} \cdot d \leq \frac{\pi}{n} \end{aligned}$$

and the robots move at most a distance of $\frac{\pi}{n}$ in each round. Therefore, it takes at least $\frac{1}{3\pi}n^2$ rounds until the radius is decreased by at least $\frac{1}{3}n$. As the circumference is 2π times the radius of a circle, decreasing the radius by $\frac{1}{3}n$ also decreases the circumference by $\frac{1}{3}n$. Thus, it takes at least $\frac{1}{3\pi}n^2$ rounds until the circumference is decreased to $\frac{2}{3}n$. □

3.5 The Upper Bound

In this section we will show that the robots gather in $\mathcal{O}(n^2)$ rounds. But before we start with the analysis, we state some well-known facts about smallest enclosing circles, on which our analysis will rely heavily.

Proposition 3.2 (Chrystal [32])**.** *Let $\mathcal{C}$ be the smallest enclosing circle (SEC) of a point set $\mathcal{S}$. Then either*

1. *there are two points $\mathsf{P}, \mathsf{Q} \in \mathcal{S}$ on the circumference of $\mathcal{C}$ such that the line segment $\overline{\mathsf{PQ}}$ is a diameter of $\mathcal{C}$, or*
2. *there are three points $\mathsf{P}, \mathsf{Q}, \mathsf{R} \in \mathcal{S}$ on the circumference of $\mathcal{C}$ such that the center c of $\mathcal{C}$ is inside $\triangle\mathsf{PQR}$, which means that $\triangle\mathsf{PQR}$ is acute-angled.*

Furthermore, the SEC of a set of points is unique.

From this proposition follows directly that the SEC of a point set P is always within the convex hull of P.

The following definition is illustrated in Figure 3.2.

Definition 3.3. Let $\mathcal{C}$ be the SEC of a set of points $\mathcal{S}$. An arc of $\mathcal{C}$ that contains no points is called a *point-free arc*. The *length* of this arc is defined as the central angle of the arc.

Note that the central angle of an arc is greater than π if the arc extends over more than half the circumference of the circle.

Proposition 3.4 (Chrystal [32])**.** *Let $\mathcal{C}$ be the SEC of a set of $n \geq 2$ points. Then there is no point-free arc with length greater than π.*

With these basics, we can now define how we measure progress. We will use two progress measures.

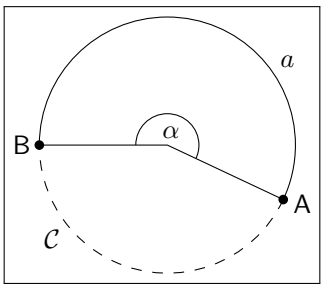


Figure 3.2: The central angle α of an arc a of the circle $\mathcal{C}$ is the angle subtended at the center of $\mathcal{C}$ by the two points A and B delimiting the arc.

- As a first progress measure, we will count the number of rounds in which robots merge. As we have n robots in the beginning, there can be at most $n-1$ such rounds.
- Since the algorithm is deterministic and it was already proven in Ando et al.'s original paper [10] that the robots gather in finite time, we know that, for a given start configuration, the point where the robots gather is fixed. We will call this point the *gathering point* M. We define a circle $\mathcal{N}_t$ with center M and radius R_t for a round t, such that $\mathcal{N}_t$ contains all robots in round t and its radius is minimal. Due to the definition of the algorithm and because the center of the SEC of a point set is always within the convex hull of the point set, the robots never leave the convex hull of their neighbors as well as the global convex hull. R_t can therefore only decrease. We will use R_t as a second progress measure.

  As the robots gather at a point inside the convex hull of the robot positions in any round t, M is inside the convex hull of the robot positions of the start configuration. Moreover, since UDG_0 is connected, the diameter of the convex hull of the robots in round 0 can be at most $n-1$ and therefore also $R_0 \leq n-1$. The idea of the proof is to show that in a constant number of rounds in which no robots merge, R_t decreases by at least $\Omega(\frac{1}{n})$.

Using these two progress measures, with $R_0 \leq n-1$ and at most $n-1$ rounds in which robots merge, it follows directly that the robots gather in $\mathcal{O}(n^2)$ rounds.

From now on, we will consider an arbitrary but fixed round t_0. Let $\mathcal{N} := \mathcal{N}_{t_0}$ and $R := R_{t_0}$. For this round, we introduce some further notions (see Figure 3.3). First, we fix an arbitrary point P on the boundary of $\mathcal{N}$ and draw a line R between P and M. Let l_2 be a line perpendicular to R such that the intersection points of l_2 and the circle $\mathcal{N}$ are in distance $\frac{1}{8}$ from P. Observe that the length of l_2 is bounded by $\frac{1}{4}$. Let l_1 be another line perpendicular to R that intersects with R halfway between P and the intersection point of l_2 with R. We define S_1 as the circular segment defined by l_1 and S_2 as the area of the segment defined by l_2 minus the area of S_1. The main idea of the analysis is to show that in round t_0 and t_0+1, either two robots merge or all robots leave S_1. We will conclude that this leads to the desired number of rounds.

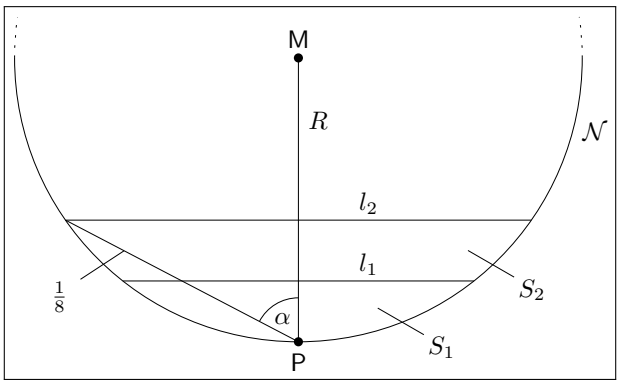


Figure 3.3: The segments S_1 and $S_1 \cup S_2$ of the global SEC are later used to measure the progress of the algorithm.

The following analysis is divided into geometric prerequisites regarding S_1 and S_2 (Section 3.5.1) and the actual analysis of the algorithm (Section 3.5.2).

3.5.1 Geometric Prerequisites

In this section we want to give prerequisites regarding S_1 and S_2 and smallest enclosing circles with centers in these segments. These will be used later to make a statement about which robots can compute target points inside one of the segments.

Lemma 3.5. *Let x be the length of a chord defining a circular segment S of $\mathcal{N}$. Then any circle $\mathcal{C}$ with its center c in S and radius $r > x$ has an arc outside of $\mathcal{N}$ with a central angle larger than π and thus cannot be the SEC of points only from $\mathcal{N}$.*

Proof. See Figure 3.4 for an illustration of the setting described by the lemma. Since r is larger than the length of the maximum distance between two points in S, both intersection points I_1 and I_2 of the circle $\mathcal{N}$ with any circle with center in S and radius $r > x$ lie outside of S. Because the center c lies in S, it follows that the (longer) arc of $\mathcal{C}$ from I_1 to I_2 outside of $\mathcal{N}$ has a central angle larger than π (the dashed part of the circumference in Figure 3.4). □

Since the chord length of $S_1 \cup S_2$ is bounded by $\frac{1}{4}$, the following corollary is immediate.

Corollary 3.6. *The radius of a SEC of a point set $\mathcal{S} \subseteq \mathcal{N}$ with its center in $S_1 \cup S_2$ is at most $\frac{1}{4}$.*

In the following, we will show two geometrical lemmas for the position of the center of a SEC, if the configuration of the underlying points adheres to a few restrictions. The first lemma follows from Corollary 3.6 and will be used to show that if a robot can see a robot that is far away from $S_1 \cup S_2$, it cannot compute a target point inside this circular segment.

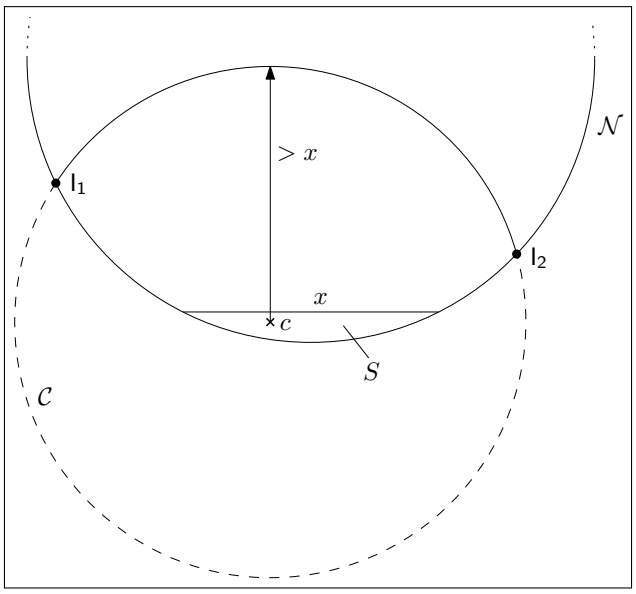


Figure 3.4: A circle with center in S and a radius exceeding the chord length of S intersects with $\mathcal{N}$ outside of S.

Lemma 3.7. *Let $\mathcal{S} \subseteq \mathcal{N}$ be a set of points. Now let A be a point in $S_1 \cup S_2$ and $\mathsf{B} \in \mathcal{S}$ be a point in distance at least 1 from A. Then the center of the SEC $\mathcal{C}$ of $\mathcal{S}$ cannot lie in the segment $S_1 \cup S_2$.*

Note that A does not need to be in $\mathcal{S}$.

Proof. Assume that $\mathcal{C}$ has its center c inside $S_1 \cup S_2$. We know from Corollary 3.6 that $\mathcal{C}$ can have at most radius $\frac{1}{4}$. Since the maximum distance of two points in $S_1 \cup S_2$ is bounded by $\frac{1}{4}$, B must have a distance of at least $\frac{3}{4}$ from $S_1 \cup S_2$ in order to be in distance at least 1 from A. Hence, B cannot lie in $\mathcal{C}$. □

The next lemma is similar to the last one in the sense that it makes a statement about configurations, for which robots cannot compute a target point in S_1. In particular, it will be used for robots that can only see one single robot in $S_1 \cup S_2$. These robots cannot compute a target point in S_1.

Lemma 3.8. *The center of the SEC $\mathcal{C}$ of a non-empty point set $\mathcal{S} \subseteq \mathcal{N} \setminus (S_1 \cup S_2)$ and a point $\mathsf{A} \in S_1 \cup S_2$ cannot lie in the segment S_1.*

Proof. Assume that $\mathcal{C}$ has its center c inside S_1. We distinguish two cases as given by Proposition 1.

1. $\mathcal{C}$ is defined by two points P_1 and P_2. A must be one of these points, say P_2, otherwise c cannot lie in S_1. Since P_1 cannot lie in S_1 or S_2 by assumption and because the height of S_1 is equal to the height of S_2, the midpoint c of $\overline{\mathsf{AP}_1}$ cannot lie in S_1.

2. $\mathcal{C}$ is defined by three points P_1, P_2 and P_3. A must be one of these points, say P_3, otherwise c cannot lie in S_1. Since $\mathcal{C}$ is the circumcircle of $\triangle \mathsf{P}_1\mathsf{P}_2\mathsf{A}$,

it lies on the intersection of the perpendicular bisectors of $\overline{\mathsf{AP}_1}$ and $\overline{\mathsf{AP}_2}$. The centers of these two segments lie outside S_1 and since the perpendicular bisectors intersect in the interior of $\triangle \mathsf{P}_1\mathsf{P}_2\mathsf{A}$ and this triangle is acute, their intersection point also cannot lie in S_1.

This completes the proof. □

Finally, as the main idea of the analysis is to show that if no robots merge, S_1 is empty after two rounds, we will need the height of S_1 to compute the progress with respect to R_t within two rounds.

Lemma 3.9. *The segment S_1 has a height h of at least $\frac{1}{128\pi \cdot R} \in \Omega\left(\frac{1}{n}\right)$.*

Proof. We start by computing the angle α (see Figure 3.3 for an illustration of α). The circumference of $\mathcal{N}$ is $2\pi R$. Thus, we can position at most $16\pi R$ points on the boundary of $\mathcal{N}$ that are in distance $\frac{1}{8}$ from the points closest to them and form a regular convex polygon. The internal angle of each of the points of this polygon is equal to 2α. To compute such an internal angle, we use that the sum of the internal angles of a convex polygon is $(m-2)\cdot\pi$, where m is the number of vertices of the polygon. In our case, this is at most $(16\pi R - 2)\cdot\pi$. It follows that each angle is at most $\frac{(16\pi R-2)\cdot\pi}{16\pi R} = \pi - \frac{1}{8R}$, and thus $\alpha \leq \frac{\pi}{2} - \frac{1}{16R}$.

Now we can use α and the fact that $\cos(x) \geq -\frac{2}{\pi}x+1$ in the interval $x \in [0, \frac{\pi}{2}]$ to compute the height h of S_1:

$$\begin{aligned} h = \frac{\cos\alpha}{16} &\geq \frac{\cos\left(\frac{\pi}{2} - \frac{1}{16R}\right)}{16} \\ &\geq \frac{1}{16}\cdot\left(-\frac{2}{\pi}\cdot\left(\frac{\pi}{2} - \frac{1}{16R}\right) + 1\right) \\ &= \frac{1}{128\pi R} \end{aligned}$$

Because $R \leq n$, we have shown $h \in \Omega(\frac{1}{n})$. □

3.5.2 Gathering Algorithm Analysis

Now we can proceed to the actual analysis of the algorithm. We can use the lemmas from Section 3.5.1 to determine that no robot can happen to compute a target point in S_1 or $S_1 \cup S_2$. Nevertheless, according to the algorithm, robots do not always reach their target point; it is also possible that they are hindered by other robots. So knowing that a target point is outside S_1 or $S_1 \cup S_2$ does not necessarily mean that the robot actually leaves the respective segment. The following two lemmas show that robots always reach their target point, if it is in $S_1 \cup S_2$, and that they cannot be hindered from leaving S_1 and S_2.

Lemma 3.10. *Robots that compute a target point in $S_1 \cup S_2$ cannot be hindered from reaching it by the limit circle of any other robot.*

Proof. Let r_i be a robot that computes a target point c (which is the center of the SEC $\mathcal{C}$) inside $S_1 \cup S_2$. Then, according to Corollary 3.6, the radius of $\mathcal{C}$ cannot exceed $\frac{1}{4}$ and thus the distance between r_i and c is also upper bounded by $\frac{1}{4}$. Now assume that there is a robot r_e that hinders r_i from reaching c. Since r_e must be a neighbor of r_i, it must also be included in $\mathcal{C}$ and therefore, r_e can have at most distance $\frac{1}{2}$ from r_i. Now let m_e be the midpoint between r_i and r_e and therefore the center of the limit circle that hinders r_i from reaching c. m_e can be at most in distance $\frac{1}{4}$ from r_i. But that means that r_i can move freely in any direction a distance of $\frac{1}{2} - \frac{1}{4} = \frac{1}{4}$ and hence it can reach its target point without being hindered by r_e. □

Lemma 3.11. *Robots cannot be hindered from leaving $S_1 \cup S_2$ by the limit circle of any other robot.*

Proof. Let r_i be a robot that computes a target point outside $S_1 \cup S_2$ in round t_0. Now assume for the sake of contradiction that there is one robot r_j that hinders r_i from leaving $S_1 \cup S_2$. This is only possible if r_j is a neighbor of r_i and thus r_j must be within distance 1 of r_i (see the circle $\mathcal{C}_1$ in Figure 3.5 with center r_i and radius 1: r_j must be in $\mathcal{C}_1$). Now let m be the point where r_i would leave $S_1 \cup S_2$ if moving to its target point. According to the algorithm it is only possible that r_i is hindered by r_j to leave $S_1 \cup S_2$, if m is not within distance $\frac{1}{2}$ from the midpoint m_j between r_i and r_j (line 6–10 of the algorithm). It follows that m_j cannot be inside the circle $\mathcal{C}_2$ (Figure 3.5) with center m and radius $\frac{1}{2}$. Based on $\mathcal{C}_2$ we can define a circle $\mathcal{C}_3$ which may not contain r_j, if m_j is not in $\mathcal{C}_2$: $\mathcal{C}_3$'s center is p_i', which is p_i reflected with respect to the point m, and its radius is 1 (see Figure 3.5). Summing up, r_j must be inside of $\mathcal{C}_1$, but outside of $\mathcal{C}_3$. Moreover, the smallest enclosing circle computed by the algorithm has at most radius 1, and so r_i's target point is at most in distance 1 of r_j. It follows that r_i's target point must be on the line between m and p_i', because each point on the straight line through p_i and m beyond p_i' is in distance more than 1 from any point that is in $\mathcal{C}_1$, but not in $\mathcal{C}_3$.

Case 1: r_j is in $S_1 \cup S_2$. Then, because the chord length of $S_1 \cup S_2$ is at most $\frac{1}{4}$, the distance between r_i and r_j is also at most $\frac{1}{4}$. But that means that r_i is at most in distance $\frac{1}{8}$ from the midpoint between r_i and r_j and thus it can move at least distance $\frac{1}{2} - \frac{1}{8} = \frac{3}{8} > \frac{1}{4}$ freely in any direction without being hindered by r_j. But after r_i has moved a distance of $\frac{1}{4}$, it has left $S_1 \cup S_2$ leading to a contradiction.

Case 2: r_j is not in $S_1 \cup S_2$. Since a SEC is defined by two or three points with at least one point on each half of the boundary of the SEC (Proposition 3.4), there must be a robot r_k that is in $S_1 \cup S_2$ and on the boundary of the SEC defining r_i's target point. It follows that r_k can be at most in distance $\frac{1}{4}$ from m. As p_i is also at most in distance $\frac{1}{4}$ from m, so is p_i' and also p_i's target point, which is between m and p_i' (see explanation above). Thus, r_k is at most in distance $\frac{1}{2}$ from r_i's target point. Since r_k is on the boundary of the SEC that defines r_i's target point, it follows that the SEC can have at most a radius of $\frac{1}{2}$. Now, since r_j is outside of $\mathcal{C}_3$ and because the distance between m and p_i' is at most $\frac{1}{4}$, r_j must be in distance greater than $\frac{1}{2}$ from r_i's target point. Thus,

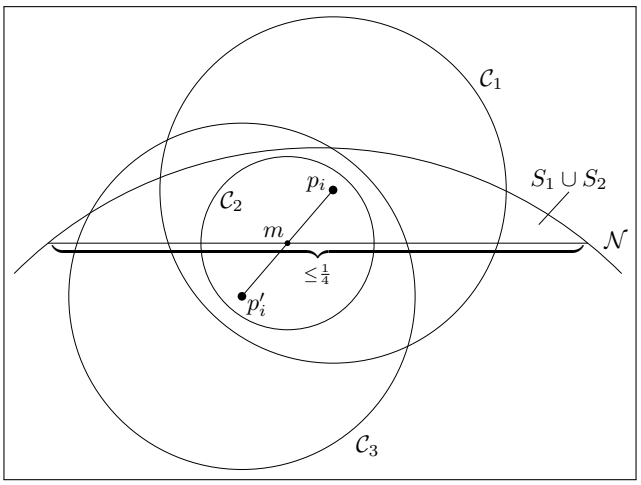


Figure 3.5: This figure illustrates the proof of Lemma 3.11. The circles indicate where r_j can be positioned: $\mathcal{C}_1$ is a circle with center p_i and radius 1 and must contain r_j. $\mathcal{C}_2$ has center m and radius $\frac{1}{2}$, and $\mathcal{C}_3$'s center is p_i' with radius 1. r_j must not be in $\mathcal{C}_3$.

r_j cannot be in the SEC that defines r_i's target point, which is a contradiction to r_i and r_j being neighbors. It follows that r_j cannot hinder r_i from leaving $S_1 \cup S_2$. □

With all these prerequisites, we can now show that if no robots merge, S_1 is empty after two rounds. We first analyze the behavior of some robots in round t_0 in Lemma 3.12, before we plug things together in Lemma 3.13.

Lemma 3.12. *Let $\mathcal{S}$ be a set of robots in round t_0 that are all positioned in or compute a target point in $S_1 \cup S_2$ and that all have a pairwise different neighborhood. Then at most one of those robots is in $S_1 \cup S_2$ at the beginning of the next round.*

Proof. Since all robots from $\mathcal{S}$ have different neighbors, there exists a robot $r_i \in \mathcal{S}$ for which no robot from $\mathcal{S}$ has a set of neighbors that is a subset of the neighbors of r_i. Thus, all robots $r_j \in \mathcal{S} \setminus \{r_i\}$ have a neighbor that is not visible from r_i and therefore in distance more than 1 from r_i. If r_i is positioned in $S_1 \cup S_2$, all robots $r_j \in \mathcal{S} \setminus \{r_i\}$ see a point B in $\mathcal{N}$ (namely the position of the neighbor that r_i cannot see) that is in distance 1 from a point A in $S_1 \cup S_2$ (namely the position of r_i). Lemma 3.7 therefore guarantees that all neighbors of r_i compute a target point outside of $S_1 \cup S_2$. According to Lemma 3.11, no robot is hindered from leaving $S_1 \cup S_2$. Thus, only r_i can stay in $S_1 \cup S_2$.

If r_i is positioned outside $S_1 \cup S_2$, it has its target point in $S_1 \cup S_2$ according to the definition of $\mathcal{S}$. Corollary 3.6 now gives that the radius of r_i's SEC cannot exceed $\frac{1}{4}$ and thus r_i is in distance at most $\frac{1}{4}$ from $S_1 \cup S_2$. Using that the distance between two points in $S_1 \cup S_2$ is at most $\frac{1}{4}$, it follows that all points within $S_1 \cup S_2$ are in distance at most $\frac{1}{2}$ from r_i. Now consider a robot $r_j \in \mathcal{S} \setminus \{r_i\}$ and a neighbor r_k of r_j that is in distance more than 1 from r_i. This robot r_k must then be in distance more than $\frac{1}{2}$ from $S_1 \cup S_2$. Since r_k is r_j's neighbor, we know

from Corollary 3.6, that the center of r_j's SEC — its target point — cannot be in $S_1 \cup S_2$ and according to Lemma 3.11 r_j is not hindered from leaving $S_1 \cup S_2$. Since this holds for all robots $r_j \in \mathcal{S} \setminus \{r_i\}$, r_i is the only robot that can be in $S_1 \cup S_2$ in round $t+1$. □

Lemma 3.13. *If $R_t \geq \frac{1}{2}$, either there are robots that merge in round t or after two rounds, the segment S_1 does not contain any robots.*

Proof. We consider all robots that are positioned in $S_1 \cup S_2$ or compute a target point in $S_1 \cup S_2$ in round t. We divide this set of robots into two subsets and analyze them separately.

- First, we consider all robots that have a neighbor with the same neighborhood. Thus, for all these robots there is another robot that computes the same target point. Then there are two possibilities: Either one of these target points is in $S_1 \cup S_2$. According to Lemma 3.10, the robots with such a target point are not hindered from reaching it and therefore they merge. If all target points are outside $S_1 \cup S_2$, Lemma 3.11 guarantees that all these robots leave $S_1 \cup S_2$.
- Now consider the robots that have a pairwise different neighborhood. According to Lemma 3.12, at most one of those robots stays in $S_1 \cup S_2$ during this round.

Thus, if r_i is positioned outside S_1 at the end of round t, we are done. Otherwise, since apart from r_i no robot is still in $S_1 \cup S_2$, we know from Lemma 3.8, that neither r_i nor a neighbor of r_i can compute a target point in S_1 in round $t+1$. Thus, r_i leaves S_1 in round $t+1$ (Lemma 3.11) and none of its neighbors enters S_1. All other robots that are not neighbors of r_i do not see a robot in S_1 and thus they cannot enter S_1. □

Lemma 3.13 will be used to show that if no robots merge, R_t decreases by $\Omega\left(\frac{1}{n}\right)$ every two rounds. According to the following Lemma, this procedure stops as soon as $R_t < \frac{1}{2}$.

Lemma 3.14 (Ando et al. [10])**.** *If $R_t < \frac{1}{2}$, the robots have gathered at one point in round $t+1$.*

This lemma holds because if $R_t < \frac{1}{2}$, all robots can see each other and thus all robots compute the same target point. It is shown Ando et al.'s original work [10] that the robots do not hinder each other from reaching this point.

Putting everything together, we are now able to prove the final result.

Theorem 3.2. *The robots gather within $\mathcal{O}(n^2)$ rounds.*

Proof. Fix an arbitrary round $t_0 \geq 0$. Since Lemma 3.13 holds for any point on the boundary of N_{t_0}, after two rounds either two robots have merged or all robots must be in distance greater than the height of S_1 from the boundary of N_{t_0}. According to Lemma 3.9, the height of S_1 is at least $\frac{1}{128 \cdot R_t}$ and thus

if the robots do not merge, the radius decreases by at least $\frac{1}{128 \cdot R_t}$, giving that $R_{t+2} \leq R_t - \frac{1}{128 \cdot R_t} \leq R_t - \frac{1}{128 \cdot R_0}$. It follows that after $2 \cdot 128 \cdot (R_0)^2 = 256 \cdot (R_0)^2$ rounds without merging robots, the radius must be less than $\frac{1}{2}$. Now it takes one round to gather the robots (Lemma 3.14). Moreover, since UDG_0 is connected, $R_0 \leq n$. There are at most $n - 1$ rounds in which robots merge. The total number of rounds is therefore at most $256 \cdot n^2 + n$. □

3.6 Conclusion

In this chapter we have shown that mobile robots can gather at a single point in $\mathcal{O}(n^2)$ rounds, when they execute the classic synchronous algorithm by Ando et al. [10]. Furthermore we showed that this bound is asymptotically tight for the algorithm by providing a matching lower bound. This raises the question whether there are more efficient algorithms.

On the other hand there are no nontrivial lower bounds known for classes of local algorithms for gathering or other formation problems. One would need a clean definition of asynchronous or synchronous local gathering strategies. A crucial property restricting such strategies is that connectivity has to be maintained. Just looking at the start configuration of the lower bound instance from Section 3.4, for example, and only demanding connectivity for this specific start configuration is not sufficient: consider the synchronous algorithm in which each point moves in the direction of the target point of our algorithm, but goes beyond this point until the distance to its neighbors is 1. This algorithm maintains connectivity for our specific start configuration, but needs only a linear number of rounds, if the start configuration positions neighboring robots in distance $\frac{2}{3}$ on the cycle. Similar results can be shown for asynchronous strategies with specific activation policies. Such examples demonstrate that the connectivity constraint has to be reflected much more severely in lower-bound models for local gathering strategies.

4

Treasure Search with Many Mobile Finite Automata

"They operate without any central control. Their collective behavior arises from local interactions." The last quote is arguably the mantra of distributed computing, however, in this case, "they" are not nodes in a distributed system; rather, this quote is taken from a biology paper that studies social insect colonies [87]. Understanding the behavior of insect colonies from a computer science perspective will hopefully prove to be a big step for both disciplines.

In this chapter, we study the process of food finding and gathering by ant colonies from a distributed computing point of view. Inspired by the model of Feinerman et al. [54], we consider a colony of n ants whose nest is located at the origin of an infinite grid. The ants collaboratively search for an adversarially hidden food source located in distance D from the origin. An ant can move between neighboring grid cells and can communicate with the ants that share the same grid cell. However, the ant's navigation and communication capabilities are very limited since its actions are controlled by a randomized *finite automaton* (FA) operating in an asynchronous environment — refer to the model section for a formal definition. Nevertheless, we design a distributed algorithm ensuring that the ants locate the food source within $\mathcal{O}(D + D^2/n)$ time units with high probability (w.h.p.)[1]. It is not difficult to show that a matching lower bound holds even under the assumptions that the ants have unbounded memory (i.e., are controlled by a Turing machine) and know the parameter n.

[1] See Section 1.1 for a formal definition.

4.1 Related Work

Feinerman et al. [53,54] introduce the aforementioned problem called *ants nearby treasure search (ANTS)* and study it, assuming that the ants (a.k.a. *agents*) are controlled by a Turing machine (with or without space bounds) and do not communicate with each other at all. They show that if the n agents know a constant approximation of n, then they can find the food source (a.k.a. *treasure*) in time $\mathcal{O}(D + D^2/n)$. Moreover, Feinerman et al. observe a matching lower bound and prove that this lower bound cannot be matched without some knowledge of n. Lenzen et al. study the effects that bounding the memory of the agents and the range of available probabilities have on the runtime [79]. In contrast to the model studied in these works, the agents in our model can communicate anywhere on the grid as long as they share the same grid cell. However, due to their weak control unit (a FA), their communication capabilities are very limited even when they do share the same grid cell. Notice that the stronger computational model of Feinerman et al. enables an individual agent to perform tasks way beyond the capabilities of a (single) agent in our setting, e.g., remember the grid cells it has already visited or perform spiral searches, which play a major role in their upper bound.

Distributed computing by finite automata has been studied in several different contexts including *population protocols* [11,15] and the recent work [51] from which we borrowed the agents communication model. In that regard, the line of work closest to our deliberations in this chapter is probably the one studying graph exploration by FA controlled agents, see, e.g., [58].

Graph exploration in general is a fundamental problem in computer science. In the typical case, the goal is for a single agent to visit all nodes in a given graph [4, 41, 43, 84, 89]. It is well-known that random walks allow a single agent to visit all nodes of a finite undirected graph in polynomial time [6]. Notice that in an infinite grid, the expected time it takes for a random walk to reach any designated cell is infinite. Our problem can also be seen as a variant of the game of cops and robbers, where the robber remains dormant [2].

Finding treasures in unknown locations has been previously studied, for example, in the context of the classic *cow-path* problem. In the typical setup, the goal is to locate a treasure on a line as quickly as possible and the performance is measured as a function of the distance to the treasure. It has been shown that there is a deterministic algorithm with a competitive ratio 9 and that a spiral search algorithm is close to optimal in the two-dimensional case [21]. The study of the cow-path problem was extended to the case of multiple agents by López-Ortiz and Sweet [80]. In their study, the agents are assumed to have unique identifiers, whereas our agents cannot be distinguished from each other (at least not at the beginning of the execution).

4.2 Model

We consider a variant of [54]'s ANTS problem, where a set of n mobile *agents* search the infinite grid for an adversarially hidden treasure. The agents are

controlled by asynchronous randomized finite automata with a common sense of direction and communicate only with agents sharing the same grid cell.

More formally, consider n mobile agents that explore $\mathbb{Z}^2$. In the beginning of the execution, all agents are positioned in a designated grid cell referred to as the *origin* (say, the cell with coordinates $(0,0) \in \mathbb{Z}^2$). We assume for simplicity that the agents can distinguish between the origin and the other cells. We denote all cells with either x or y-coordinate being 0 as *north/east/south/west-axis*, depending on their location.

The main difference between our variation of the ANTS model and the original one lies in the agents' computation and communication capabilities. In both variants, all agents run the same (randomized) protocol. However, under the model considered in this chapter, the agents are controlled by an asynchronous randomized *finite automaton*. This means that the individual agent has a constant memory and thus, in general, can store neither coordinates in $\mathbb{Z}^2$ nor the number of agents. On the other hand, in contrast to the model considered in [54], our agents may communicate with each other. Specifically, under our model, an agent a positioned in cell $c \in \mathbb{Z}^2$ can communicate with all other agents positioned in cell c at the same time. This communication is quite limited though: agent a merely senses for each state q of the finite automaton, whether there exists at least one agent $a' \neq a$ in cell c whose current state is q. Notice that this communication scheme is a special case of the one-two-many communication scheme introduced in [51] with bounding parameter $b = 1$.

The *distance* between two grid cells $(x,y),(x',y') \in \mathbb{Z}^2$ is defined with respect to the ℓ_1 norm (a.k.a. Manhattan distance), that is, $|x-x'|+|y-y'|$. Two cells are called *neighbors* if the distance between them is 1. In each step of the execution, agent a positioned in cell $(x,y) \in \mathbb{Z}^2$ can either move to one of the four neighboring cells $(x,y+1),(x,y-1),(x+1,y),(x-1,y)$, or stay put in cell (x,y). The former four *position transitions* are denoted by the corresponding cardinal directions N,S,E,W, whereas the latter (stationary) position transition is denoted by P (standing for "stay put"). We recall that the agents have a common sense of orientation, i.e., the cardinal directions are aligned with the corresponding grid axes for every agent in every cell.

The agents operate in an asynchronous environment. Each agent's execution progresses in discrete (asynchronous) steps indexed by the non-negative integers and we denote the time at which agent a completed step $i > 0$ by $t_a(i) > 0$. Following the common practice, we assume that the time stamps $t_a(i)$ are determined by the policy ψ of an adversary that knows the protocol but is oblivious to its random bits, whereas the agents do not have any sense of time. The set of activation times determined by the adversary is called a *schedule* and we will use the terms synchronous/asynchronous policy and -/- schedule interchangeably in the rest of the chapter, despite their subtle difference.

Formally, the agents' protocol is captured by the 3-tuple $\Pi = \langle Q, s_0, \delta \rangle$, where Q is the finite set of *states*; $s_0 \in Q$ is the *initial state*; and

$$\delta : Q \times 2^Q \to 2^{Q \times \{N,S,E,W,P\}}$$

is the *transition function*. At time 0, all agents are in state s_0 and positioned in

the origin. Suppose that at time $t_a(i)$, agent a is in state $q \in Q$ and positioned in cell $c \in \mathbb{Z}^2$. Then, the state $q' \in Q$ of a at time $t_a(i+1)$ and its corresponding position transition $\tau \in \{N, S, E, W, P\}$ are dictated based on the transition function δ by picking the pair $(q', \tau) \in \delta(q, Q_a)$, uniformly at random from $\delta(q, Q_a)$, where $Q_a \subseteq Q$ contains state $p \in Q$ if and only if there exists some (at least one) agent $a' \neq a$ such that a' is in state p and positioned in cell c at time $t_a(i)$. (Step i is deterministic if $|\delta(q, Q_a)| \leq 1$.) For simplicity, we assume that while the state subset Q_a (input to δ) is determined based on the status of cell c at time $t_a(i)$, the actual application of the transition function δ occurs instantaneously at the end of the step, i.e., agent a is considered to be in state q and positioned in cell c throughout the time interval $[t_a(i), t_a(i+1))$.

4.2.1 Problem Setting

The goal of the agents is to locate an adversarially hidden *treasure*, i.e., to bring at least one agent to the cell in which the treasure is positioned. The distance of the treasure from the origin is denoted by D. As in [54], we measure the performance of a protocol in terms of its run-time, where the time is scaled so that $t_a(i+1) - t_a(i) \leq 1$ for every agent a and step $i \geq 0$. Although we express the run-time complexity in terms of the parameters n and D, we point out that neither of these two parameters is known to the agents (who cannot even store them in their very limited memory).

4.3 Parallel Diamond Search

In this section, we introduce the collaborative search strategy DiamondSearch that depends on an *emission scheme*, which divides all participating agents in the origin into *teams* of size ten and emits these teams continuously from the origin until all search teams have been emitted. We delay the description of our emission scheme until Section 4.4 and describe for now the general search strategy (without a concrete emission scheme). We assume, for the sake of the following informal explanation, an environment in which the agents operate in synchronous rounds and then explain how we can lift this assumption.

The DiamondSearch strategy consists of two stages. The first stage works as follows: Whenever a team is emitted, one agent becomes an *explorer* and four agents become *guides*, one for each cardinal direction. The remaining five agents become *scouts*, whose function will be explained later. Each guide walks into its respective cardinal direction until it hits the first cell that is not occupied by another guide. The explorer follows the north-guide and when they hit the non-occupied cell $(0, d) \in \mathbb{Z}^2$ for some $d > 0$, the explorer starts a *diamond search* by first walking south-west towards the west-guide. When it hits a guide, the explorer changes direction to south-east, then to north-east, and finally to north-west. This way, it traverses all cells in distance d from the origin, referred to hereafter as *level* d (and also almost all cells in distance $d+1$). When the explorer meets a guide on its way, the guide enters a *sleep* state to be awoken again in

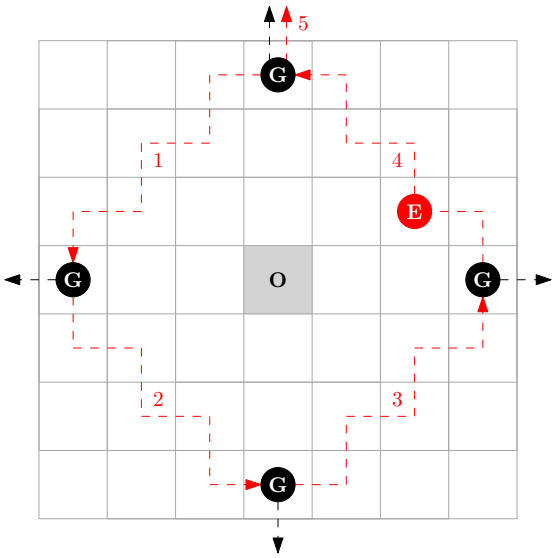


Figure 4.1: An explorer (E) starts a diamond search in level $\ell = 3$ at the north-guide, visits all guides (G) in level ℓ in a counter-clockwise fashion and ends at the north-guide. Whenever the explorer meets a guide, the guide moves outwards in its cardinal direction. When the explorer completes its search by arriving again at the north-guide, both agents walk outwards together to the next level to be searched. (The numbers indicate the order in which the explorer moves.)

the second stage. The explorer also enters a sleep state after arriving again at the north-guide, thereby completing the first stage of the diamond search.

The second stage of DiamondSearch is started when the last search team is emitted from the origin. At this point in time, $\Theta(n)$ cells are occupied by sleeping guides/explorers in all four cardinal directions. The last search team wakes up the innermost sleeping search team upon which it resumes its job and walks outwards to explore the next unexplored level in the same way as in the first stage. Each team recursively wakes up the search team of the next level until all sleeping teams have been woken up and resumed the search. A search in the second stage has one important difference in comparison to a search in the first stage: When an explorer meets a guide g during a search, g moves outwards to the next unexplored level instead of entering a sleep state, hopping over all the other stationary guides on its way, and waits there for the next appearance of an explorer. When the explorer has finished its diamond by reaching the north-guide again, it moves north (with the north-guide) to the first unexplored level and starts another diamond search there. Knowing that all other guides have reached their target positions in the same level as well, a new search can begin. Figure 4.1 gives an illustration of the process for a single search team.

Note that the (temporary) assumption of a synchronous environment is crucial for the correctness of the algorithm described so far as we assume that whenever an explorer crosses a coordinate axis, the respective cell contains a guide. In an asynchronous setting, the guide might still be on its way to that particular cell and hence, the explorer would continue walking diagonally ad in-

finitum. We counter this problem by coupling the searches for different levels in such a way that a search in level ℓ can never have progressed further than a search in level $\ell' < \ell$. This implies that a search in level ℓ cannot start/finish earlier than a search in level $\ell' < \ell$ starts/finishes. This coupling is implemented by equipping each explorer with a *scout* that essentially allows the explorer e_ℓ in level ℓ to check whether the explorer $e_{\ell-1}$ of the preceding level has already progressed at least as far as e_ℓ and to move only then. On top of that, explorer e_ℓ only leaves a coordinate axis after ensuring, again by means of its scout, that there is already a guide present in level $\ell + 1$. This additional check together with a few technicalities described later suffices to ensure that the searches are "nested" properly and the corresponding guides of each explorer are waiting in the right positions along the coordinate axes when those are hit by the explorer.

As a (much desirable) byproduct of the aforementioned explorers' logic, it is guaranteed that during the execution of the DiamondSearch strategy, every cell contains at most one explorer of each possible state. To ensure that the same holds for the guides, they are also equipped with scouts whose role is to check that during a guide's journey outwards, it does not move into a cell which is already occupied by a guide, unless the latter is in a stationary state (waiting for its explorer).

4.3.1 The DiamondSearch Strategy

After the preceding overview description of the DiamondSearch strategy, we shall now specify it in greater detail and precision in order to allow us to formally prove important properties later on.

Emission scheme. Initially, all n agents are located at the origin. Until all agents become involved in the DiamondSearch strategy, an *emission scheme* is responsible for emitting new teams (each consisting of ten agents) from the origin. The emission of the teams is spaced apart in time in the sense that no two teams are emitted at the exact same time. (Under a synchronous schedule, a spacing of 20 time units is guaranteed.) To formally express (and analyze) the emission rate, we introduce the notion of an *emission function* $f_n : \mathbb{Z}_0^+ \to \mathbb{Z}_0^+$, where, until all teams are emitted, $f_n(t)$ bounds from below the number of teams emitted up to time t. For simplicity, we assume that there are enough agents to execute our algorithms, i.e., that $n \geq 40$.

Let $k + 2$, $k \in \Theta(n)$, be the total number of emitted teams where we assume $k \geq 2$. The first and last emitted teams have a special role as *signal teams* in our protocol. The k remaining teams $s_1, \ldots, s_k$ will be referred to as *search teams*. Whenever a search team becomes ready, four of the ten agents become MovingGuides — one for each cardinal direction — and walk outwards in their corresponding directions, while the fifth one becomes a MovingExplorer and follows the north-MovingGuide (see below for a detailed description of the agent types). Each MovingGuide and MovingExplorer is accompanied by a Scout that will stick to this particular agent for the rest of the execution.

Agent types. In the remainder of this chapter, we will refer to several different types of agents. Since there is only a constant number of different types, these can be modeled by having individual finite automata for the various types. We essentially use six different types and explain their specific behavior in the following: Scout, Guide, MovingGuide, MovingExplorer, WaitingExplorer, and Explorer. We will use the terms "outwards" and "inwards" in the context of agents of the two Guide-types (recall that they are associated with a cardinal direction) to indicate the respective direction away from or towards the origin. We subsume the types Explorer/MovingExplorer/WaitingExplorer and Guide/MovingGuide under the name *explorers* and *guides*, resp. During the process of the algorithm, each non-Scout agent will be accompanied by a Scout, whose type is specific to the type of the agent it is accompanying — its *owner*. Since all different Scout-types have very similar tasks, we first give a general description of a Scout's function and then explain its type-specific behavior together with the owner's behavior.

Scout. The function of each Scout-type is to control when its owner is allowed to move further. It does so by moving to one of the four neighbor cells of the owner — the *scout position* — and waiting for a certain condition (the presence/absence of a certain type of agent) to become true in that cell. When the condition is met, the Scout moves back to the cell containing its owner and notifies the owner. When the owner moves to a new position, the Scout moves along. As Scouts only play an auxiliary role in our protocol, we may refer to a cell as *empty* even if it contains Scouts. The owner of a Scout only performs an action after knowing that the Scout is in the correct position.

Guide. A Guide waits until a Explorer performing a search (this can be encoded in the state of the Explorer) has entered its cell. When (1) the cell one coordinate inwards is empty and (2) its cell contains neither MovingGuide nor MovingExplorer, it becomes a MovingGuide.

MovingGuide. A MovingGuide moves outwards (at least one cell) until it hits a cell c that contains no Guide. The north-MovingGuide moves north together with the MovingExplorer of the same search team. It does so by verifying before each move (after the first) that the MovingExplorer has caught up and is in the same cell. Otherwise, it waits for the MovingExplorer to catch up. Upon arriving in c, the MovingGuide becomes a Guide, and waits for an Explorer to visit. A MovingGuide uses its Scout to prevent moving in a cell that contains a MovingGuide, MovingExplorer or Explorer.

MovingExplorer. A MovingExplorer repeatedly moves north together with the north-MovingGuide of the same search team. More precisely, it only moves north when there is no MovingGuide in its cell, implying that the MovingGuide is already one cell further and waits there for the MovingExplorer to catch up. The MovingExplorer moves until it hits the first cell c that contains neither a Guide that already has an Explorer searching (this can be encoded in the state of the

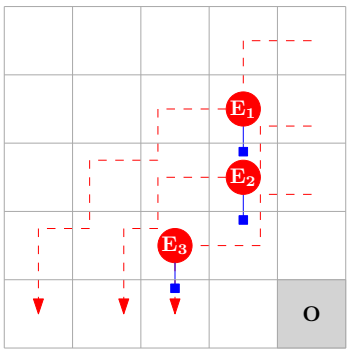


Figure 4.2: Three Explorers (E_1 to E_3) are performing a search of adjacent levels in the north-west quadrant. Their Scouts are depicted as blue squares connected to their respective Explorer. E_1 cannot walk further as there is still an Explorer (E_2) in the cell checked by its Scout. Both E_2 and E_3 can walk further as their Scouts do not observe an explorer in the checked cells.

Guide) nor an Explorer. As soon as cell c contains a Guide (the north-Guide of this explorer's team), it becomes an Explorer. A MovingExplorer uses its Scout to prevent moving into a cell that contains another MovingExplorer while walking outwards.

WaitingExplorer. A WaitingExplorer waits until its cell is empty and then becomes a MovingExplorer.

Explorer. An Explorer does the bulk of the actual search by moving along the sides of a diamond using Guides on its way to change direction (see Figure 4.1 for an illustration). In the process, it moves south-west, then south-east, north-east, and north-west, in this order. Initially, an Explorer performs one move west and then alternatingly south and west.

During a diagonal walk, an Explorer uses its Scout to prevent it from overtaking Explorers closer to the origin during their search as follows. Consider an Explorer e in the north-west quarter-plane (walking south-west). The Scout is sent to the south-neighbor cell, the *scout position*, and notifies e, when no Explorer is present there (which might immediately be the case). Only then, the Explorer and the Scout move one cell further where the Scout again enters the scout position. See Figure 4.2 for an illustration of a diagonal walk in the north-west quadrant and in particular of the function of the Scouts. In the south-west, south-east, and north-east quarter-planes, the scout positions are the east-, north-, and west-, neighbor cells, resp.

When the Explorer meets a west/south/east-Guide in an axis cell c, it changes its moving direction. Before leaving the axis, it waits until c does contain neither Guide nor MovingGuide (thereby ensuring that there is a Guide one cell outwards). Upon arrival back at the north Guide after the diamond search is completed, it becomes a WaitingExplorer.

The Explorer of the search team exploring level 1 counts its steps (the exploration journey at this level contains exactly 8 cells) and uses the Scout to make

sure that the cells on the coordinate axes contain a Guide before entering them.

The signal teams. The first and last emitted teams, s_0 and s_{k+1}, resp., have a special role and they do not actively participate in the exploration of the grid (which is handled by $s_1, \ldots, s_k$). Their job is solely to signal to the other teams when the second stage of the protocol begins.

The first team s_0 enters a special *signal* state and stays at the origin until the last team s_{k+1} has been emitted. (Due to the design of our emission scheme in Section 4.4, the agents in team s_{k+1} know that they belong to the last emitted team and are able to notify the agents of s_0 accordingly.) The aforementioned logic of the agents in DiamondSearch ensures that as long as there is an agent present in the origin, the Guides and Explorers of the innermost search team (and recursively all other search teams) cannot move outwards. When s_0 is notified by s_{k+1}, the agents in both teams switch to a designated idle state, ignored by all other agents. As now the origin appears to be empty, the Guides and Explorers of the innermost (and eventually the other search teams) can move outwards to continue searching — the second stage has begun.

4.3.2 Correctness

In this section we establish the correctness of the DiamondSearch strategy by proving that each cell is eventually explored and no agent is lost in the process. We say that a cell in level ℓ is *explored* after it has been visited by an Explorer exploring level ℓ, where we recall that level $\ell \in \mathbb{Z}_0^+$ consists of all cells in distance ℓ from the origin. An Explorer is said to *start* a (diamond) search in level ℓ at time t if it moves west from the cell $(0, \ell)$ (containing the north Guide) at time t and it *finishes* a (diamond) search in level ℓ at time t if it enters the cell $(0, \ell)$ from the east at time t. The *start time* t_ℓ^S, *finish time* t_ℓ^F, and *move time* t_ℓ^M are given by the times at which an Explorer starts a search in level ℓ, finishes a search in level ℓ, and when the WaitingExplorer in level ℓ becomes a MovingExplorer, resp. An Explorer *explores* level ℓ at time t, if $t_\ell^S < t < t_\ell^F$. The design of DiamondSearch ensures that regardless of the emission scheme used, the Guides in every cardinal direction occupy a contiguous segment of cells. It also implies the following observation and lemma.

Observation 4.1. *For two levels $\ell' > \ell$, we have $t_{\ell'}^S > t_\ell^S$, $t_{\ell'}^F > t_\ell^F$, and $t_{\ell'}^M > t_\ell^M$.*

Proof. Consider some level $\ell > 1$. An Explorer can only start a search in level ℓ by leaving cell $(0, \ell)$ after the Explorer of level $\ell - 1$ has left cell $(0, \ell - 1)$, which implies $t_\ell^S > t_{\ell-1}^S$. An Explorer in level ℓ can only move to cell $(0, \ell)$ and thereby finish the search in level ℓ after the Explorer of level $\ell - 1$ has already reached the cell $(0, \ell - 1)$, thus $t_\ell^F > t_{\ell-1}^F$. A WaitingExplorer in level ℓ can become a MovingExplorer only when its Guide has left cell $(0, \ell)$. This requires in turn that the Explorer in level $\ell - 1$ has already left cell $(0, \ell - 1)$ which requires that it already has become a MovingExplorer, hence $t_\ell^M > t_{\ell-1}^M$. □

Lemma 4.2. *Outside the origin, no two agents of the same type occupy the same cell at the same time.*

Proof. First, recall that the MovingExplorers emitted from the origin are emitted at different times. A WaitingExplorer becomes a MovingExplorer only when there is no other MovingExplorer in the same cell. MovingExplorers use Scouts to prevent stepping onto each others' cells. A MovingExplorer becomes an Explorer in cell c only if c does not contain an Explorer and Explorers use Scouts to prevent stepping into each others' cells. As no two Explorers can be in the same cell, neither can be two WaitingExplorers and the claim for all explorer agents follows.

MovingGuides are emitted from the origin at different times. A Guide becomes a MovingGuide only if its cell does not contain a MovingGuide. MovingGuides use Scouts to prevent stepping onto each others' cells. A MovingGuide becomes a Guide only if its cell does not contain a Guide, which establishes the assertion for the guides.

As the Scouts of owners with different types also have different types, we only need to show the claim for Scouts of the same type. This follows by labeling a Scout as north/east/south/west-CheckingScout, while it is checking a scout condition in the north/east/south/west neighboring cell of its owner's cell. □

Each Explorer relies on Guides to indicate when it has to change the search direction in order to search a specific level. The next lemma gives a guarantee for this.

Lemma 4.3. *Whenever an Explorer enters a cell c on an axis, cell c contains a Guide.*

Proof. Observe that if c lies on the north-axis, it will contain a Guide when the Explorer e returns there because e only leaves c to start a search when c contains a Guide and this Guide stays there until e returns. To prove the claim for the other axis, let s_ℓ be the search team exploring level ℓ and let e_ℓ be the corresponding explorer. We prove the statement by induction on ℓ. Observe that the statement holds for e_1 as the Explorer of the first search team explicitly counts cells and uses its Scout to ensure that c contains a Guide.

Consider a cell c_ℓ on an axis in level ℓ and assume as the induction hypothesis that the cell $c_{\ell-1}$ on the axis in level $\ell-1$ contained a Guide when it was entered by Explorer $e_{\ell-1}$. As $e_{\ell-1}$ blocks e_ℓ from overtaking it, Explorer e_ℓ can enter c_ℓ only after $e_{\ell-1}$ has left $c_{\ell-1}$, which $e_{\ell-1}$ only does after ensuring that $c_{\ell-1}$ is empty. This requires, in turn, that the Guide in cell $c_{\ell-1}$ has become a MovingGuide and left $c_{\ell-1}$. This can only happen after all other MovingGuides have passed $c_{\ell-1}$ and the MovingGuide in cell c_ℓ has become a Guide, thereby asserting the claim. □

The canonical paths. In what follows, we use paths in the infinite grid in their usual graph-theoretic sense, viewing a path p as a (finite or infinite) sequence of cells, where $p(i)$ and $p(i+1)$ are grid neighbors for every $i \geq 1$. Notice that unless stated otherwise, the paths mentioned are not necessarily simple.

Let $s_1, \ldots, s_k$ be the search teams emitted from the origin (ignoring the two signal teams s_0 and s_{k+1}) ordered by ascending emission time and consider some agent a participating in one of the search teams $s_1, \ldots, s_k$. Given some

adversarial policy ψ, let p_a^ψ be the path traversed by a during the execution of the algorithm under ψ starting at the time at which a is emitted from the origin. We extend the sequence defined by p_a^ψ, fixing $p_a^\psi(0) = (0, 0)$. We shall refer to p_a^ψ as the *execution path* of a (under ψ).

The logic of the guides directly implies that if agent a is a north/south/east/west guide, then its execution path satisfies $p_a^\psi(i) = (0, i)/(0, -i)/(i, 0)/(-i, 0)$ for every adversarial policy ψ. In other words, the path traversed by a guide does not depend on the adversarial policy. We argue that this is in fact the case for all agent types, introducing the notion of a *canonical path*.

Lemma 4.4. *For every $1 \leq i \leq k$ and for each agent role ρ (among the 10 different roles in a search team), there exists a canonical path $p_{i,\rho}^*$ such that if agent a is the ρ-agent in search team s_i, then $p_a^\psi = p_{i,\rho}^*$, regardless of the adversarial policy ψ.*

Proof. As noted above, the assertion holds for the guides. Since the execution path of a scout is fully determined by the execution path of its owner, it suffices to show that the assertion holds for the explorer e_i of search team s_i.

Let $m_{i,j}$ be the simple path leading from cell $(0, i)$ to cell $(0, j-1)$ along the north axis and let r_ℓ be the simple path corresponding to the diamond search of level ℓ, starting in cell $(0, \ell)$ and ending in cell $(1, \ell)$, that is, the path traversed by an explorer exploring level ℓ during the time interval $[t_\ell^S, t_\ell^F)$. We argue that the execution path of explorer e_i is always

$$p_{i,e}^* = m_{0,i} \circ r_i \circ m_{i,k+i} \circ r_{k+i} \circ m_{k+i,2k+i} \circ r_{2k+i} \cdots ,$$

namely, agent e_i (and search team s_i) search levels $\ell = z \cdot k + i$ for $z = 0, 1, \ldots$, where we recall that k is the number of search teams.

To establish this argument, we prove that if e_i currently searches level ℓ, then the next level it is going to search is $\ell+k$. The argument then follows by induction on z as the first level searched by e_i (after it is emitted from the origin) is level i. To that end, consider the explorer e_i at time t_ℓ^F when it is back in cell $(0, \ell)$ as an Explorer. Recall that e_i becomes a MovingExplorer and starts moving outwards at time t_ℓ^M that occurs only after the corresponding north-Guide has become a MovingGuide and left cell $(0, \ell)$. This is turn happens only after cell $(0, \ell-1)$ was verified as empty, which implies that at time t_ℓ^M, every other MovingExplorer e_j is positioned in some cell $(0, \ell')$, $\ell' > \ell$. Since a MovingExplorer does not enter a cell containing another MovingExplorer, it follows that e_i will not overtake e_j as long as both are MovingExplorers, but rather pass over its north-Guide after e_j has already started exploring its next level. Therefore, explorer e_i will have to pass the north-Guides of all other search teams before it gets to its next explored level, which completes the proof as there are k search teams in total. □

It will sometimes be convenient to use the notation p_a^* for the canonical path $p_{i,\rho}^*$ when agent a is the ρ-agent of search team s_i. The key to Lemma 4.4's proof is the observation that since MovingExplorers do not overtake each other, the explorers maintain a *cyclic order* between them in terms of the levels they explore. The exact same argument can be applied to the guides, concluding that the agents of a search team "stick together" throughout the execution.

Corollary 4.5. *The agents that were emitted from the origin as guides of search team s_i serve as* Guides *in levels $\ell = z \cdot k + i$ for $z = 0, 1, \ldots$*

Preventing dead/live-locks. We now turn to prove that DiamondSearch does not run into deadlocks. Recall that during the execution of DiamondSearch, agents often wait for other agents to complete some task before they can proceed. In particular, we say that agent a is *delayed* by agent a' at time t, denoted $a \rightarrow_t a'$, if at time t, a is positioned in some cell c and resides in some state q and the DiamondSearch strategy dictates that a can neither leave cell c nor move to any state other than q until a' performs some action in cell c that may take the form of entering cell c, leaving cell c, or moving to some state within cell c. For example, a guide in an axis cell c is delayed by its corresponding explorer until the latter reaches c. Another example is an explorer which is delayed by its scout in some north-west quarter-plane cell (x, y), while the latter is delayed until the explorer exploring the previous level leaves cell $(x, y - 1)$. To avoid the necessity to account for the scouts, we extend the definition of delays in the context of the correctness proof, allowing for agent a in cell c to be delayed by agent a' in a neighboring cell c' if a is actually delayed by its scout in c who is delayed by a' in c'.

Let $\mathcal{D}_t$ be the directed graph that corresponds to the binary relation $\rightarrow_t$ over the set of agents. We prove that DiamondSearch does not run into deadlocks by establishing the following lemma.

Lemma 4.6. *The directed graph $\mathcal{D}_t$ does not admit any (directed) cycle at all times t.*

Proof. Consider a snapshot of the agents' states and positions at time t. Examining the DiamondSearch strategy, one realizes that the outermost MovingExplorer and MovingGuides are not delayed by any other agent and that the i^{th} outermost MovingExplorer and MovingGuides can only be delayed by the $(i-1)^{\text{th}}$ outermost MovingExplorer and MovingGuides. The innermost Explorer e is not delayed by any agent as long as it is not in an axis cell. In an axis cell, e can only be delayed by the corresponding guide. An innermost guide in cell c is delayed by its corresponding explorer until the latter reaches cell c and since then, it can only be delayed by the corresponding innermost MovingGuide. Non-innermost Explorer and Guides in level ℓ can only be delayed by the Explorer and Guides in level $\ell - 1$ or by the MovingExplorers and MovingGuides. The assertion follows. □

The following corollary is derived from Lemma 4.6 since there is a constant number of state transitions an agent positioned in cell c can perform before it leaves cell c.

Corollary 4.7. *Agent a reaches cell $p_a^*(i)$ within finite time for every $i \geq 1$.*

Since the canonical path p_a^* contains infinitely many different nodes for every agent a, we can deduce from Corollary 4.7 that DiamondSearch does not run into livelocks, thus establishing the following theorem.

Theorem 4.1. *The cell containing the treasure is explored in finite time.*

4.3.3 Runtime Analysis

For the sake of a clearer run-time analysis, we analyze DiamondSearch employing an ideal emission scheme with emission function $f_n(t) = \Omega(t)$, i.e., a new search team is emitted from the origin every constant number of time units. We do not know how to implement such a scheme, but in Section 4.4, we will describe an emission scheme with an almost ideal emission function of $f_n(t) = \Omega(t - \log n)$ and in Section 4.5, we will show how to compensate for the gap.

Our proof consists of two parts. First, we analyze the run-time of DiamondSearch assuming a "synchronous" adversarial policy ψ^{s}, where $t_a(i) = i$ for all a and i. Then, we lift this assumption by showing that ψ^{s} is actually the worst case policy. We start with the following lemmas.

Lemma 4.8. *Under ψ^{s}, we have $t^M_{\ell+1} - t^M_\ell \geq 4$ and $t^S_{\ell+1} - t^S_\ell \geq 4$.*

Proof. By Observation 4.1, we know that a search in level $\ell + 1$ cannot finish before a search in level ℓ. By construction of the algorithm, $e_{\ell+1}$ cannot become a MovingExplorer in level $\ell + 1$ before time $t^M_\ell + 4$, hence $t^M_{\ell+1} - t^M_\ell \geq 4$.

Consider the two explorers $e_{\ell+1}$ and e_ℓ exploring levels $\ell+1$ and ℓ, resp. The design of the emission process and the inequality $t^M_{\ell+1} - t^M_\ell \geq 4$ imply that when $e_{\ell+1}$ and e_ℓ were MovingExplorers, they had a distance of at least 3 with $e_{\ell+1}$ being closer to the origin. As $e_{\ell+1}$ has to walk to level $\ell + 1$ to start a search and e_ℓ only to level ℓ, the claim holds. □

Lemma 4.9. *Under ψ^{s}, the explorer of search team s_i is not delayed after time t^M_i.*

Proof. Recall that t^M_i is the time when the Explorer of search team s_i turns into a MovingExplorer after search level i in the first stage of the algorithm. By Lemma 4.8, we know that after time t^M_i, the distance between any two MovingExplorers at least 3 and hence, they cannot delay each other. It is easy to see that under ψ^{s}, an Explorer never has to wait for the Guide of the next level in order to leave an axis and thus cannot be delayed by a Guide. Since $t^S_{\ell+1} - t^S_\ell \geq 4$, two Explorers of adjacent levels can never delay each other either. □

Lemma 4.10. *Under ψ^{s}, we have $t^F_\ell \in \mathcal{O}(\ell + \ell^2/n)$ for any level $\ell > 0$.*

Proof. Consider level $\ell \leq k$ and recall that this level will be searched by the ℓ^{th} search team s_ℓ and that k is the number of search teams. Let t_ℓ be the time at which s_ℓ is emitted from the origin and note that f_n guarantees $t_\ell \in \mathcal{O}(\ell)$. Observe that no delays can occur during the first stage. Hence, s_ℓ reaches level ℓ after time $\mathcal{O}(\ell)$, visits the 8ℓ cells to explore level ℓ in time $\mathcal{O}(\ell)$ and thus guarantees $t^F_\ell \in \mathcal{O}(\ell)$. Moreover, since the last team is emitted from the origin by time $\mathcal{O}(k)$ and at this time, all search teams are positioned in the first k levels, it follows that each search team s_i starts its second stage by time $\mathcal{O}(k)$, that is, $t^M_i = \mathcal{O}(k)$.

Consider now level $\ell > k$. Assume wlog. that level ℓ is explored by search team i and let e be the explorer of that search team. Lemma 4.4 guarantees that e moves along its canonical path p^*_e. Let π be the canonical path p^*_e truncated after

the exploration of level ℓ. Combining Lemma 4.9 with the fact that $t_i^M = \mathcal{O}(k)$, and recalling that $\ell > k = \Theta(n)$, it suffices to show that the length of π (in hops) is $|\pi| = \mathcal{O}(\ell^2/k)$.

To that end, we write $|\pi| = m_m + m_x$, where m_m is the number of hops in π that e performs as a MovingExplorer and m_x is the number of hops in π it performs as an Explorer. The design of DiamondSearch ensures that $m_m = \mathcal{O}(\ell)$ which is $\mathcal{O}(\ell^2/k)$ as $\ell > k$. Since e is part of s_i, we know that $m_x = \sum_{z=0}^{\lfloor \ell/k \rfloor} 8(i + zk) = \mathcal{O}(\ell^2/k)$, which yields the assertion. □

We now turn to show that the run-time of DiamondSearch under any adversarial policy ψ is at most the run-time under ψ^{s}. By definition, policy ψ^{s} maximizes the length of the time between consecutive completion times of the agents' steps. Informally, we have to prove that by speeding up some agents, the adversary cannot cause larger delays later on.

To that end, consider two agents a and a' and recall that Lemma 4.4 guarantees that they follow the canonical paths p_a^* and $p_{a'}^*$, resp., regardless of the adversarial policy. The agents can delay each other only when they are in the same cell, so suppose that there exist two indices i and i' such that $p_a^*(i) = p_{a'}^*(i') = c$.

Given some adversarial policy ψ, let $t_{\mathrm{in}}^{\psi}(a)$ (resp., $t_{\mathrm{in}}^{\psi}(a')$) be the time at which agent a (resp., a') enters c in the step corresponding to $p_a^*(i)$ (resp., $p_{a'}^*(i')$) under ψ and let $t_{\mathrm{out}}^{\psi}(a)$ (resp., $t_{\mathrm{out}}^{\psi}(a')$) be the time at which agent a (resp., a') exits c for the first time following $t_{\mathrm{in}}^{\psi}(a)$ (resp., $t_{\mathrm{in}}^{\psi}(a')$) under ψ. The key observation now is that the adversarial policy does not affect the order in which a and a' enter/exit cell c.

Observation 4.11. *For every two adversarial policies ψ_1, ψ_2, we have $t_{in}^{\psi_1}(a) < t_{in}^{\psi_1}(a')$ if and only if $t_{in}^{\psi_2}(a) < t_{in}^{\psi_2}(a')$ and $t_{out}^{\psi_1}(a) < t_{out}^{\psi_1}(a')$ if and only if $t_{out}^{\psi_2}(a) < t_{out}^{\psi_2}(a')$.*

Therefore, the adversary may decide to modify its policy relatively to ψ^{s} by speeding up some steps of some agents, but this modification cannot delay the progression of the agents along their canonical paths. Corollary 4.12 now follows from Lemma 4.10.

Corollary 4.12. *Under any adversarial policy, $t_\ell^F \in \mathcal{O}(\ell + \ell^2/n)$ for any level $\ell > 0$.*

4.4 An Almost Optimal Emission Scheme

We introduce the emission scheme *ParallelTeamAssignment* that w.h.p. guarantees an emission function of $f_n(t) = \Omega(t - \log n)$. In Section 4.5, we describe the search strategy GeometricSearch, that yields an optimal run-time of $\mathcal{O}(D+D^2/n)$ when combined with DiamondSearch. The main goal of this section is to establish the following theorem.

Theorem 4.2. *Employing the ParallelTeamAssignment emission scheme, DiamondSearch locates the treasure in time $\mathcal{O}(D + D^2/n + \log n)$ w.h.p.*

Our first goal is to describe the process FastSpread, where n agents spread out along the east-axis R consisting of the cells $(x, 0)$ for $x \in \mathbb{Z}^+$ such that each cell in some prefix of R is eventually assigned to a single agent. The main idea behind the implementation of FastSpread is that on every step, agent a throws a fair coin and moves outwards (towards east) if the coin shows heads and stays put otherwise. If a senses that it is the only agent occupying cell c, then it marks itself as *ready* and stops moving; cell c is also said to be *ready* following this event. Furthermore, when a walks onto a ready cell, it moves outwards deterministically.

To prevent any cell from becoming empty, the agents employ a mechanism that ensures that at least one agent stays put in each cell. To implement this mechanism, the agents decide in advance, i.e., in step i, if they want to move in step $i + 1$ and report their decision to the other agents. In other words, an agent a throws a coin in step i and enters a state H or T that correspond to throwing heads or tails, resp. Then, a moves outwards in step $i + 1$ if and only if it entered state H in step i and if it senses at least one other agent in state T. Informally, a only moves if at least one other agent has promised to stay put the next time it acts.

Next, we show that the protocol works correctly, i.e., no cell in the prefix of R will become empty before getting ready. Suppose for contradiction that there is a cell c, such that c becomes empty at time t. Let a be an agent and i a step of a such that for all agents a' in cell c and all steps j, it holds that $t_{a'}(j) \leq t_a(i) < t$. In other words, no agent in c changes its state during time $t_a(i) < t' < t$. According to the design of our protocol, a must sense some other agent a' in state T precisely at time $t_a(i)$. Since a' does not wake up after $t_a(i)$ and before t, it follows that a' resides in state T at time t, which is a contradiction.

Lemma 4.13. *For every positive integer $s \leq 16n$, the first $s/16$ cells of the ray R are ready after $s + \mathcal{O}(\log n)$ time units w.h.p.*

Proof. Let X_a be the random variable that counts the number of moves a non-ready agent a made outwards by time $s \leq 16n$. Unfortunately, the moves that a makes are not independent of the previous moves. Therefore, we study a weaker probabilistic process, where the number of a's moves dominates X_a. Assume that a occupies cell c at time $t_a(j)$ and let $a_0, a_1, \ldots, a_{z-1}$ denote the non-ready agents that occupy cell c at time $t_a(j)$. In the weaker process, $a = a_i$ only moves in step $j+1$ if a_{i+1} (index arithmetic in this proof is modulo z) is in state T and a is in state H at time $t_a(j)$ or if there is a ready agent in c.

Let j' be the last step a_{i+1} performs before $t_a(j)$. The probability that a_{i+1} enters state T in step j' is $1/2$. In addition, the probability that a_i enters H in step j is $1/2$ and therefore, the probability of a_i actually moving towards east in step $j + 1$ is at least $1/4$ in the weaker process. Since we want to count movements of a_i that are independent of the previous movements, we divide the execution of FastSpread into intervals of two time units. Now the last step that a_i executes in the end of each such interval only depends on the previous step that a_{i+1} executed. Since every agent wakes up at least once per time unit, both

of these steps are unique for every interval and, in particular, independent of the previous time intervals.

Let X'_a be the random variable that counts the number of moves a made east in the weaker process conditioned on the event that a is not ready by time $s + \mathcal{O}(\log n)$. Since the coin tosses made in each step are independent and a moves towards east every two time units with probability at least $1/4$, we get that $E[X'_a] \geq (1/2) \cdot (1/4) \cdot (s + \mathcal{O}(\log n))$. By applying a Chernoff bound we get that $P(X'_a < 1/2 \cdot E[X'_a]) \in \mathcal{O}(n^{-h})$ for an arbitrary constant $h > 0$. Since $X_a \geq X'_a$, any agent a that is not ready by time $s + \mathcal{O}(\log n)$, the distance to the origin is at least $s/16$ w.h.p. □

Intuitively, the aforementioned process can be seen as parallel leader election. Since we want to describe an efficient emission scheme, it remains to show how the process can be used to quickly emit search teams consisting of five agents with their respective Scouts from the origin. To enable the FastSpread procedure to elect ten different kinds of agents per search team, we dedicate every tenth cell to a specific kind of agent. As an example, every cell in distance $d \equiv 1 \pmod{10}$ is dedicated to an Explorer. After an Explorer is alone in a cell using the FastSpread procedure described above, it collects its search team in the following manner: it first takes one step east where a leader election for the Scout dedicated to it takes place. If the corresponding cell is occupied by a Scout that is marked ready, they both move outwards to collect the next agent. Otherwise, the Explorer waits until the leader election is over. After the Explorer (accompanied by the collected Guides and Scouts) collected all agents needed for the search team, the team walks to the origin from where it will then be emitted into the four cardinal directions. We refer to the FastSpread protocol combined with the collection of the agents as ParallelTeamAssignment.

In addition, we always keep track of the innermost search team in the following way. We flag the agents in the leftmost cell that has not been collected as the innermost agents. Every time an agent moves out of the innermost cell with a coin toss, this flag is turned off. In addition, when the Explorer collects its search team, it performs one additional move outwards to flag the cell where the Explorer for the next team is elected as the innermost. An Explorer only starts collecting its search team after it has been flagged as the innermost agent. This way we know that the closer the team is elected to the origin, the earlier it starts moving towards the origin. The use of the grid by the ParallelTeamAssignment is illustrated in Figure 4.3.

Similarly as in the DiamondSearch protocol, every agent a always checks with its Scout before moving that the next cell is not occupied by an agent of the same type to prevent a from overtaking any of the other agents. Given that the agents never overtake or pass other agents, we observe that the canonical path p^*_a for any agent is fixed, i.e., is independent of the adversarial policy, starting from the time when a becomes ready. Furthermore, under ψ^{s}, all agents in a single team enter the origin simultaneously and ParallelTeamAssignment provides a spacing of more than 6 between emitted teams.

Lemma 4.14. *Assume that n agents start executing ParallelTeamAssignment*

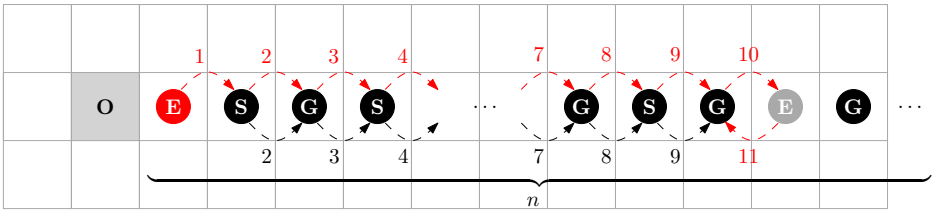


Figure 4.3: First, the n agents executing the ParallelTeamAssignment protocol form a ray of single agents in cells $(0,1),\dots(0,n)$. After the innermost Explorer e (E, red) in cell $(0,1)$ (denoted in red) is ready, it starts collecting its search team of Guides (G) and Scouts (S). Assuming that all the agents of e's search team are ready, e has collected all the agents after at most ten time units. Then in at most two more time units, e flags the Explorer of the next team (E, gray) as innermost.

protocol in round 0. *Then at least* $\lfloor \min\{s,n\}/10 \rfloor$ *search teams have entered the origin by time* $17s + \mathcal{O}(\log n)$ *w.h.p.*

Proof. By Lemma 4.13, the first s cells are ready w.h.p. by time $t' = 16s + \mathcal{O}(\log n)$, which indicates that the agents in these cells are ready to perform their collection process latest at time t'. In addition, in each time unit after t', the Explorers occupying one of the first s cells moves east unless they have already collected their teams. Therefore, latest at time $16s + \mathcal{O}(\log n) + 10 = t$, all full teams within the first s cells have been collected after which they start moving towards the origin.

Let $a_1,\dots,a_m$ denote the non-Scout agents of some type, say Explorers, within the first s cells. We first observe that after a_i has moved $10i$ times after being collected, it reaches the origin. Next, we point out that obeying a synchronous schedule, no agent ever gets blocked by the other agents. Assume that agent a_i is blocked by a_{i-1} from entering cell c. This indicates that a_i has made more than 1 move per time unit. Furthermore, a_i is able to move to c without blocks latest when it would have moved to c according to the synchronous schedule.

It follows that all agents from any team e_i reaches the origin latest when they would reach the origin according to the synchronous schedule. Since performing $10i$ moves takes $10i$ time units in the synchronous schedule, all agents in team e_m will reach the origin latest at time $t + 10m \le t' + s \le 17s + \mathcal{O}(\log n)$. □

By Lemma 4.14, the emission function $f_n(t)$ provided by the ParallelTeamAssignment protocol satisfies $f_n(t) = \Omega(t - \log n)$ and therefore, Theorem 4.2 follows.

4.5 Optimal Diamond Search

In this section, we will present the search strategy *HybridSearch* that locates the treasure with optimal run-time of $\mathcal{O}(D + D^2/n)$. This is achieved by, combining DiamondSearch employing the ParallelTeamAssignment with the randomized search strategy GeometricSearch that is fast only if the treasure is close to the origin.

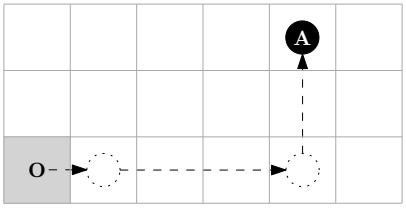


Figure 4.4: An agent A performs a GeometricSearch in the north-east quarter-plane. It first moves one cell to the east, then a geometrically distributed number of steps to the east followed by a geometrically distributed number of steps towards north. The dashed circles show the intermediate positions of the agents after the different stages.

The search strategy GeometricSearch is suited to locate the treasure very quickly if it is located close to the origin, more precisely if $D \leq \log(n)/2$. Initially, each of the n agents chooses uniformly at random one of the four quarter-planes that it will be searching. We will explain the strategy exemplary for an agent "responsible" for the north-east quarter-plane. The other three types operate analogously in their respective quarter-plane.

Initially, the agent moves one cell to the east. From then on, it moves a geometrically distributed number of steps east following which it moves a geometrically distributed number of steps to the north. More precisely, with probability 1/2 the agent moves further and otherwise stops walking in the current direction. Both these processes can be realized in our model by having two state transitions where one of them moves the agent further while the other one ends the current walk. Either of the two transitions is chosen uniformly at random and a walk of geometrically distributed length is obtained. See Figure 4.4 for an illustration of such a search.

Lemma 4.15. *If $D \leq \log(n)/2$, then GeometricSearch locates the treasure in time $\mathcal{O}(D)$ w.h.p.*

Proof. Consider some cell c at distance $d \leq \log(n)/2$ from the origin and fix some agent a. Let X_a be a random variable that captures the length of the walk of agent a and observe that X_a obeys a negative binomial distribution so that

$$P(X_a = k) = (k+1) \cdot 2^{-(k+2)} \ .$$

Recalling that a has already moved one step, we conclude that the probability that a moves up to distance d is

$$P(X_a = d-1) = d \cdot 2^{-d-1} \geq 2^{-d-1} = \Omega(1/\sqrt{n}) \ .$$

Since all cells at distance d from the root have the same probability of being explored by a and since there are $\mathcal{O}(\log n)$ such cells, it follows that a explores cell c with probability at least $\Omega\left(\frac{1}{\sqrt{n}\log n}\right)$. Therefore, the probability that none

of the agents explores cell c is at most

$$\left(1 - \Omega\left(\frac{1}{\sqrt{n}\log n}\right)\right)^n < e^{-\Omega\left(\frac{\sqrt{n}}{\log n}\right)}.$$

The assertion follows. □

We can now combine the two search strategies GeometricSearch, which is optimal for $D \leq \log(n)/2$, and DiamondSearch employing ParallelTeamAssignment, which is optimal for $D = \Omega(\log n)$, into the HybridSearch strategy as follows.

At the beginning of the execution, each agent tosses a fair coin to decide whether it participates in DiamondSearch or GeometricSearch. Let n_r and n_g be the number of agents participating in DiamondSearch and GeometricSearch, resp. and observe that $n_r, n_g \geq n/3$ w.h.p. Then the agents enter according states so that they do not interfere with each other anymore. One group executes GeometricSearch and locates the treasure w.h.p. in time $\mathcal{O}(D)$ if $D \leq \log(n)/2$ and the other group executes DiamondSearch locates the treasure w.h.p. in time $\mathcal{O}(D + D^2/n)$ if $D = \Omega(\log n)$, thereby establishing Theorem 4.3.

Theorem 4.3. *HybridSearch locates the treasure in time $\mathcal{O}(D + D^2/n)$ w.h.p.*

4.6 Conclusion

One of the main motivations of distributed computing researchers is to find problems in which separated computationally powerful devices can be replaced by many collaborative computationally weaker entities without deteriorating performance guarantees. In this chapter, we indeed added another example to the long list of instances in which this principle has been demonstrated to hold. Combined with the lower bound of Feinerman et al., our result demonstrates that by allowing the agents to use a very primitive means of communication, one can get rid of the requirement for a super-constant memory and an approximation of n and still obtain the same (optimal) runtime for locating the treasure.

It is important to point out that our algorithm is not inspired by any observations regarding the real behavior of ants. As such, we do not claim that our results explain any natural phenomenon, but rather attempt to advance the understanding of the power and limitations of a basic nature-inspired model. In that regard, the tightness of our upper bound indicates that there is no point in attempts to improve the lower bound or alternatively, that other restrictions on the agents should be considered.

5

Treasure Search with Few Mobile Finite Automata

Recent research (and in particular the previous chapter) on understanding the behavior of insect colonies from a distributed computing perspective has mainly focused on questions like "How long does it take a large collection of ants to locate a food source?" [50,54] or "How do the computational capabilities of a single ant within this collection affect the time until the food source is found?" [53,58,79].

In this chapter, we take a computability point of view and, instead of focusing on *large* numbers of agents and on the time required to find a food source, analyze the *minimum* number of agents that is sufficient to locate a food source within (expected) finite time. More precisely, we show that the minimally sufficient number of agents crucially depends on the model assumptions, i.e., whether each agent is controlled by a *finite automaton* (FA) or a *pushdown automaton* (PDA), whether it has access to random bits or not, and whether the environment is synchronous or asynchronous.[1] For most combinations of the aforementioned characteristics, we establish lower and upper bounds on the number of agents required to locate the food. Our bounds are tight in most cases. We essentially present two different families of algorithms — rectangle/spiral and geometric searches — are inspired by our previous results [50]. The main contributions of this chapter, however, are the lower bounds for two deterministic FA-agents and one deterministic PDA-agent presented in Sections 5.4.1 and 5.5.2, respectively. Table 5.1 at the end of the chapter gives a complete picture of our findings.

As border cases of our findings, we point out that in an asynchronous setting

[1] Notice the striking resemblance to the problem of finding the number of people needed to change a light bulb: For people, the answer usually depends on nationality and profession while for ants, it depends on timing and computational power.

four agents are sufficient to solve the problem when their computational capabilities are most restricted, i.e., they are controlled by deterministic FAs. If we allow access to random bits and grant the agents slightly more computational power — a PDA — already one single agent can solve the problem. Note that neither of these results require the full computational power of a Turing machine.

We do not claim that our considerations are particularly relevant from a biological perspective — an ant hive generally consists of significantly more than four ants. However, our results show that powerful computational capabilities can be traded for primitive means of communication while still being able to solve complex problems — even for small number of agents.

5.1 Model

The model that we consider in this chapter is mostly identical to the one presented in Section 4.2 in the previous chapter. In the following, we briefly recap the previous model while highlighting the few modifications that we require for our deliberations in this chapter. For a full description of the underlying model, we refer the reader to Section 4.2.

As in Chapter 4, we consider n mobile agents with a common sense of direction exploring the infinite integer grid in an asynchronous or synchronous environment. In contrast to previous work (and in particular the previous chapter), we do *not* assume that the agents can distinguish between the origin and the other cells (see Section 5.1.3 for a discussion of this matter). Furthermore, the agents are controlled by a finite automaton (as in the previous chapter) or by a pushdown automaton, both either randomized or deterministic. An agent a in some cell c can communicate with the other agents in cell c by sensing for each state q of its (finite or pushdown) automaton whether there exists at least one other agent $a' \neq a$ in cell c whose current state is q.

Since we only consider instances with a constant number of agents, we allow each agent to run a different individual protocol. This is modeled by assigning to each agent an individual initial state in the respective automaton (note that this is only relevant in the deterministic case as otherwise coin flips can be used to break symmetry). The protocol is controlled by either a finite automaton or a pushdown automaton. We shall first explain the semantics of the former and then explain the additional capabilities of the latter.

FA-protocol. An agent controlled by a *FA-protocol* behaves exactly like the agents described in the previous section with the additional possibility of a different initial state per agent, which allows the agents to perform different tasks also in the absence of randomization. Hence, the agent's protocol is captured by the 3-tuple $\Pi = \langle Q, s_0^a, \delta \rangle$, where Q is the finite set of *states*, $s_0^a \in Q$ is the *initial state* of agent a, and $\delta : Q \times 2^Q \to 2^{Q \times \{N,S,E,W,P\}}$ is the *transition function*.

PDA-protocol. An agent employing a *PDA-protocol* essentially operates identically but is allowed to use a pushdown automaton with an infinite stack

instead of an FA. In each step, the agent reads and removes the top-most symbol from the stack ("pop") — if the stack is empty, the agent reads the special symbol ε and the stack remains unchanged — and then adds a finite amount of symbols to the top of the stack ("push"). The symbol read from the stack serves as additional input to the agent. Formally, the agents' protocol is captured by the 4-tuple $\Pi = \langle Q, s_0^a, \Gamma, \delta \rangle$, where Q is the finite set of *states*, $s_0^a \in Q$ is the *initial state* of agent a, Γ is the finite *stack alphabet*, and $\delta : Q \times 2^Q \times \Gamma \cup \{\varepsilon\} \to 2^{Q \times \Gamma^* \times \{N,E,S,W,P\}}$ is the *transition function*. Suppose that at time $t_a(i)$, agent a is in state $q \in Q$, positioned in cell $c \in \mathbb{Z}^2$, and the top-most symbol on the stack is $\gamma \in \Gamma \cup \{\varepsilon\}$. Then, the state $q' \in Q$ of agent a at time $t_a(i+1)$, the word $\alpha \in \Gamma^*$ to be written to the stack, and the corresponding movement $\tau \in \{N, E, S, W, P\}$ are dictated based on the transition function δ by picking the tuple (q', α, τ) uniformly at random from $\delta(q, \gamma, Q_a)$, where $Q_a \subseteq Q$ contains state $p \in Q$ if and only if there exists some (at least one) agent $a' \neq a$ such that a' is in state p and positioned in cell c at time $t_a(i)$.

As before, we assume that while Q_a (input to δ) is determined based on the status of cell c at time $t_a(i)$, the actual application of the transition function δ occurs instantaneously at the end of the step, i.e., agent a is considered to be in state q and positioned in cell c throughout the time interval $[t_a(i), t_a(i+1))$.

5.1.1 Problem Setting

We consider two different variants of the problem, where the goal in both is to locate an adversarially hidden *treasure*, i.e., to bring at least one agent to the cell in which the treasure is positioned while the distance of the treasure from the origin is denoted by D. In async-ANTS, the problem is to find the treasure in an arbitrary asynchronous environment while in the sync-ANTS problem the agents operate in a synchronous environment. A FA/PDA-protocol $\mathcal{P}$ is *effective* if it allows the agents to locate the treasure in finite time if $\mathcal{P}$ is deterministic, or if the agents locate the treasure in expected finite time if $\mathcal{P}$ is randomized.

5.1.2 Preliminaries

For our deliberations we require a set of definitions. Let $\mathcal{A}$ be the set of agents. We denote by $E_a^{\mathcal{P}}(t)$ the cells that an agent a employing protocol $\mathcal{P}$ has visited until time t and furthermore $E^{\mathcal{P}}(t) = \bigcup_{a \in \mathcal{A}} E_a^{\mathcal{P}}(t)$. In the context of the sync-ANTS problem, we take the liberty to write $E_a^{\mathcal{P}}(i)$ for a (then global) step i as shorthand for $E_a^{\mathcal{P}}(t_a(i))$ and analogous for $E^{\mathcal{P}}(i)$. We omit $\mathcal{P}$ in the previous expressions if the considered protocol is clear from the context.

5.1.3 Non-Distinguishable Origin

As mentioned above, the agents *cannot* distinguish the origin from the other cells on the grid in the current model. This is in clear contrast to most previous work, where the origin usually is a special cell that can be observed by the agents. Here we will briefly justify why we deviate from the established literature in this respect.

The goal of this chapter is to determine the minimum number of agents required to locate the treasure under different model variations. Previous work mainly considered large numbers of ants where a distinguishable origin can be emulated easily by leaving a single dedicated agent at the origin without affecting the asymptotic parameters of the algorithm. Clearly, such an assumption does not effectively change the power of a model when one is interested in asymptotics.

Since we consider small, i.e., constant, number of agents, the situation is slightly different. As we present algorithms that employ one agent solely to mark the origin, one could argue analogously to the case of many agents and allow the agents to distinguish the origin and in return raise the minimum number of required agents by one. However, we also present algorithms that do not use an agent to mark the origin and therefore would not benefit from such a modification. It seems, hence, that a distinguishable origin is not an essential requirement for effective algorithms and thus we decided to consider the weaker model, thereby leaving it to the discretion of the algorithm designer whether or not a distinguishable origin is needed.

5.2 Four Agents

The goal of this section is to solve the async-ANTS problem without using randomization. We provide a simple protocol for four FA-agents that uses three of the four agents as landmarks for the fourth agent. The fourth agent discovers the whole grid in a spiraling fashion with increasing distance to the origin.

We begin by giving an informal description of the protocol. The landmark agents, referred to as *guides*, position themselves in a triangle around the origin and after getting a signal from the searching agent, called the *explorer*, move step by step further away from the origin. The explorer moves to the guides one by one signaling them to expand the triangle. This way the explorer is able to guarantee that it can always reach one guide after meeting another by simply walking a (possibly diagonal) straight line, even after the guides are within a super-constant distance from each other and the origin.

All three guides have specific roles and therefore we give them task-specific names: NorthGuide, WestGuide and EastGuide. The agents execute the following protocol, which is illustrated in Figure 5.1. The protocol is initialized by the NorthGuide moving once north, the WestGuide moving once west and the EastGuide moving once east. After the explorer notices that the origin is empty, it moves once north. Note that we refer to the general role of an agent (explorer/guide) with a normal font while the specific type is marked with a sans-serif font (e.g., Explorer or WaitingExplorer).

NorthGuide. When the NorthGuide meets a WaitingExplorer, it moves once north.

WestGuide. When the WestGuide meets a WaitingExplorer it moves once west and becomes a MovingWestGuide. The MovingWestGuide first moves once west and then once south and becomes a WestGuide again.

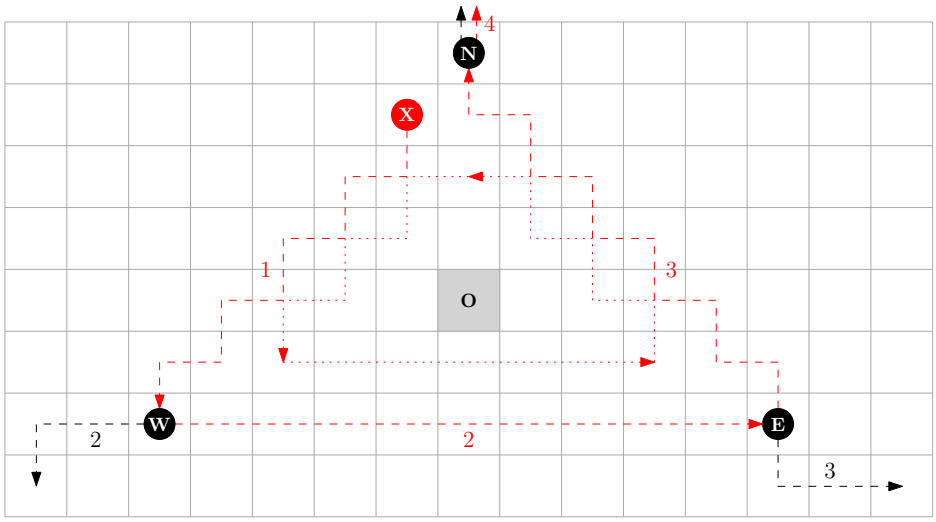


Figure 5.1: Four agents are discovering the grid and currently are performing a triangle search in distance 3. The origin is denoted by a gray square, the Explorer (X) by a red disk and the NorthGuide (N), WestGuide (W) and EastGuide (E) by black disks labeled with the corresponding initial letters. The dashed lines indicate the paths that the agents are moving along whereas the numbers indicate the order of the movements; moves along the arrow labeled with i are performed only after the moves along the arrow labeled with $i-1$ are finished. The dotted red line indicates the path of the Explorer in distance 2.

EastGuide. When the EastGuide meets a WaitingExplorer it moves once south and becomes a MovingEastGuide. The MovingEastGuide moves twice east and becomes again an EastGuide.

Explorer. The Explorer continuously performs *triangle searches* in increasing distances. It continuously moves into a given direction, starting with south-west (by alternatingly moving south and west). When the Explorer meets a WestGuide, it changes its moving direction to east and becomes a WaitingExplorer. When it meets an EastGuide, it changes the direction to north-west and becomes a WaitingExplorer. Finally, when the Explorer meets a NorthGuide, it changes its moving direction to south-west (alternates between west and south) and becomes a WaitingExplorer. Notice that the Explorer meets the NorthGuide in the starting position of the triangle search in the next distance. Whenever the Explorer meets a MovingWestGuide or a MovingEastGuide in cell c, it waits until c is empty before continuing to move.

WaitingExplorer. When the WaitingExplorer resides in a cell that does not contain an EastGuide, a NorthGuide, or a WestGuide, it becomes an Explorer and continues moving.

We index the triangle searches by their distances, i.e., if the Explorer meets the NorthGuide in cell $(0, i)$ and starts moving south-west, we index the corresponding triangle search by index i (observe the similarity to the concept of a level used

in the previous chapter) and denote it by TS_i. A triangle search in distance i starts when the Explorer leaves cell $(0, i)$ by moving west and ends when the Explorer meets a NorthGuide. Furthermore, we say that TS_i *works correctly*, if the Explorer meets the WestGuide only in cell $(-2i + 1, -i + 1)$, the EastGuide only in cell $(2i - 1, -i + 1)$ and the NorthGuide only in cell $(0, i + 1)$ during TS_i.

Lemma 5.1. *Every triangle search works correctly.*

Proof. We prove the statement by induction on the distance of the triangle searches. Consider TS_1 as the base case. Initially, all the Guides are located in cells adjacent to the origin. By the design of our protocol, the Explorer first makes sure that the NorthGuide goes into cell $(0, 2)$. After this, it moves south-west and reaches the WestGuide in cell $(-1, 0)$. Then it travels east and reaches the EastGuide in cell $(1, 0)$. From there, it travels north-west and meets the NorthGuide in cell $(0, 2)$. Thus, the claim holds for TS_1.

Assume then that the claim holds for TS_{i-1} and consider TS_i. The Explorer starts moving south-west from cell $(0, i)$. According to the induction assumption, the WestGuide is located in either $(-2i + 2, -i + 2)$, $(-2i + 1, -i + 2)$ or $(-2i + 1, -i + 1)$. Since the Explorer moves diagonally, it has to pass all of these cells. According to the design of our algorithm, it does not overtake the MovingWestGuide, i.e., the MovingWestGuide reaches its destination before the Explorer, and therefore the Explorer meets the WestGuide in cell $(-2i + 1, -i + 1)$.

Similarly, when the Explorer starts moving towards east, the correctness of the previous triangle ensures that the MovingEastGuide reaches the cell $(2i - 1, -i + 1)$ before the Explorer. After meeting the EastGuide, the Explorer starts moving diagonally towards the starting point and reaches it after $2i$ movements. Since the Explorer moves north in the next step, it meets the NorthGuide in cell $(0, i + 1)$. □

To show that the treasure eventually gets discovered, we need two more auxiliary observations. First, we show that every cell in distance d is discovered latest during TS_{d+1}. Second, we show that each triangle search finishes within finite time. We call the set of cells along which the Explorer moves during TS_i the path of triangle search i.

Observation 5.2. *Every cell c within distance d to the origin is discovered latest during TS_{d+1}.*

Proof. We prove the claim by induction on the distances of the cells, i.e., we show that all cells within distance d are contained in a triangle search with index at most $d + 1$. The base case is clear since the origin is contained within the path that the Explorer moves during TS_1.

Assume then that the claim holds for all cells in distance d. By the design of the triangle search protocol, the path of TS_{i+1} contains all the cells adjacent to the cells in the path of TS_i that are not discovered during TS_i. See Figure 5.1 for an illustration. Therefore, all cells in distance $d + 1$ are discovered latest during TS_{d+2}. □

Observation 5.3. *Every triangle search ends within finite time.*

Proof. Let t be the time when TS_i starts for some $i > 0$. By Lemma 5.1, we know that TS_{i-1} worked correctly and therefore we know that the WestGuide reaches cell $(-2i+1, -i+1)$ and the EastGuide reaches cell $(2i-1, -i+1)$ latest by time $t+3$. Therefore, latest by time $t+3+4i$, the Explorer meets the WestGuide in cell $(-2i+1, -i+1)$. By time $t+3+4i+2$, the WestGuide has left the cell and the Explorer can continue moving east. By time $t+5+4i+4i+2$, the Explorer turns towards the NorthGuide and finally reaches its cell by time $t+7+8i+4i$ ending the triangle search. □

We can now combine the results from this section. Recall that D is the distance to the treasure. By Observation 5.2, the treasure is found latest during TS_{D+1}. As the duration of each search is finite by Observation 5.3 and by Lemma 5.1 each triangle is eventually searched, we get the following theorem.

Theorem 5.1. *There exists an effective deterministic FA-protocol for async-ANTS for $n = 4$.*

5.3 Three Agents

In this section, we show that we can locate the treasure with three deterministic agents if they operate in a synchronous environment and that in an asynchronous environment, three randomized agents are sufficient.

5.3.1 Deterministic Protocol for sync-ANTS

We show that we can get rid of one of the FA-agents from the algorithm presented in Section 5.2 by giving the agents a common notion of time. In other words, if we assume that the execution of the algorithm is synchronous, three agents suffice to discover the treasure. Our goal is to prove the following theorem.

Theorem 5.2. *There exists an effective deterministic FA-protocol for sync-ANTS for $n = 3$.*

The idea of the three-agent protocol is similar to the protocol from Section 5.2 and inspired by the DiamondSearch protocol presented in the previous chapter. Again, one of the agents, the Explorer, performs the actual searching and the two other agents work as Guides. The task of one of the Guides, called OriginGuide, is simply to stand still and mark the origin throughout the execution. The task of the other Guide is to tell the Explorer when it hits an axis. On the first round of the execution, the Explorer and the other Guide move one step north to cell $(0, 1)$ and then start the execution of the following protocol.

Explorer. The Explorer repeatedly performs *diamond searches* in increasing distances. It starts the first diamond search in distance 1 by diagonally moving south-west, i.e., alternating between moving west and south. When it meets a Guide, it alters its movement direction by 90° counter-clockwise. At the end of a complete diamond (i.e., when meeting a Guide again at the starting point), it

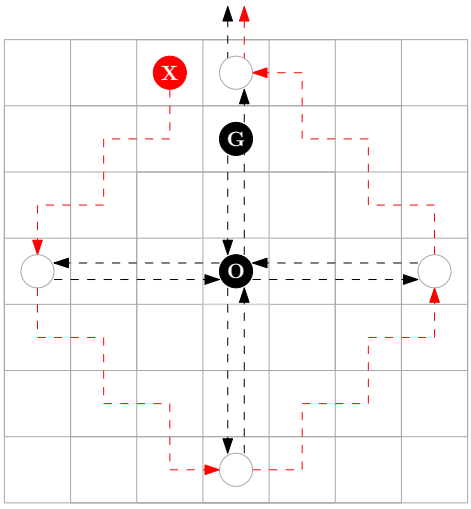


Figure 5.2: Three agents can discover the entire grid under a synchronous environment. The gray circles indicate the locations where the Explorer (X) meets the Guide (G). The OriginGuide (O) marks the origin.

moves one step outwards starting a new diamond search with a larger distance. During a diamond search in distance d, the Explorer discovers all cells that have distance d to the origin.

Guide. The Guide starts by moving towards the OriginGuide that marks the origin. When it meets the OriginGuide, it alters its direction by 90° clockwise and moves outwards. When it meets the Explorer, it turns around and moves inwards towards the OriginGuide. The Guide also moves one step north with the Explorer when they meet in the end of searching a diamond and starts walking towards the OriginGuide afterwards.

The execution of our protocol is illustrated in Figure 5.2. To prove Theorem 5.2, we only need to show that every time the Explorer enters a cell on an axis, it meets a Guide. To see why this is sufficient, consider any cell c on the plane with distance d to the origin. Then c is searched (latest) during diamond search in distance d. Therefore, assuming that each diamond search is performed correctly, the whole plane is eventually discovered.

It is fairly easy to see that the Explorer and the Guide never fail to meet. Consider round r when the Explorer and a Guide meet on an axis during diamond search in distance d. Then the distance that both of them have to move until the next meeting point is $2d$. Since both agents move exactly once per round, the claim follows. Note that the assumption of a synchronous environment is crucial here.

5.3.2 Randomized Protocol for async-ANTS

We now show that if we are not restricted to deterministic state machines but allow randomization, we can find the treasure under an asynchronous environment

with only three FA-agents by using a protocol inspired by the GeometricSearch strategy presented in the previous chapter.

Again, we have two Guides and one Explorer and the task of one of the agents, the OriginGuide, is to simply stay in the origin. The Explorer performs the actual searching and starts by uniformly at random choosing either (north, east), (east, south), (south, west) or (west, north), i.e., it randomly chooses a quarter plane. Then, the Explorer performs a *geometric search* on that quarter-plane.

Consider the case of choosing (east, south); the other quarter-planes work analogously. The Guide and the Explorer execute the following protocols.

Explorer. The Explorer starts by moving once east. Then on every step the Explorer tosses a fair coin and if it shows heads, it moves east. When the coin shows tail, the Explorer stops and becomes a WaitingExplorer until its cell is occupied by a WaitingGuide. When the WaitingGuide appears, the WaitingExplorer moves one cell south, becomes an Explorer, and continues tossing coins but now moves one cell south every time the coin shows head instead of east. When the coin shows tails, the Explorer turns back, i.e., starts moving north. After the Explorer reaches a cell with a WaitingGuide, it stops and moves west (until it reaches an OriginGuide) whenever its cell contains no WaitingGuide.

Guide. The Guide moves east on every step if its cell is not occupied by an Explorer. When it meets a WaitingExplorer, it turns into a WaitingGuide. When the WaitingGuide meets an Explorer, it becomes a Guide again and moves west whenever its cell is not occupied by an Explorer until it meets an OriginGuide.

After all the agents reach the origin, they restart the process. The protocol is illustrated in Figure 5.3. It is easy to see that each geometric search has a finite duration with probability 1 since the Explorer throws a finite number of heads in every search with probability 1. Assume that the number of heads is finite. Then the Explorer becomes a WaitingExplorer in finite time. After the Explorer becomes a WaitingExplorer, the Guide moves towards the cell of the WaitingExplorer in every step and therefore reaches it in finite time. Similarly, the Explorer returns to the WaitingGuide in finite time and they both reach the OriginGuide in finite time.

Theorem 5.3. *There exists an effective randomized FA-protocol for* async-ANTS *for* $n = 3$.

Proof. Assume that the treasure is located in cell $c = (x, y)$ in the north-east quarter plane with $D = x + y$. Let us index the geometric searches, i.e., the iterations of the algorithm, by the positive integers. Clearly, the protocol is defined so that if the treasure is found in search i, then search $j > i$ is not needed, however, for the sake of the analysis, we assume that the agents keep performing the searches indefinitely and bound the time until the treasure is found — let T be the random variable that captures this time. Given this view, we know that search i is independent of all searches other than i.

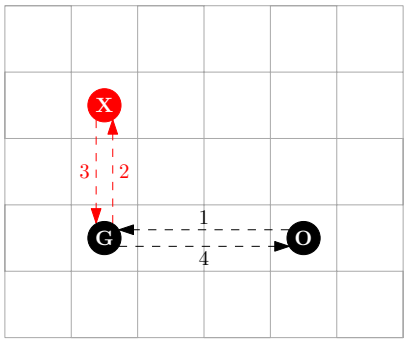


Figure 5.3: Three agents are performing a geometric search on the north-west quarter plane. Moves along the black arrows are executed by both the Explorer (X) and the Guide (G) while the OriginGuide (O) states at the origin. Moves along the red arrows are executed only by the Explorer.

Let A_i be the event that the Explorer finds the treasure in search i. This happens if it chooses the right quarter plane, throws heads exactly $x-1$ times before throwing tails once and then throws heads $y-1$ times. Hence, $\Pr(A_i) = \frac{1}{4} \cdot 2^{-(x-1)} \cdot \frac{1}{2} \cdot 2^{-(y-1)} = 2^{-(D+1)}$. Let $B_i = \neg A_1 \wedge \cdots \neg A_{i-1} \wedge A_i$ be the event that the treasure is found in search i and not in any search $j < i$. Let L_i be the random variable that measures the number of distinct cells the Explorer visits in search i, i.e., the length of the path along which the Explorer moves during search i. We rely on the following equations that hold for every $i \geq 1$ and $1 \leq j < i$:

(1) $\Pr(A_i) = 2^{-(D+1)}$

(2) $\Pr(B_i) = (1 - 2^{-(D+1)})^{i-1} 2^{-(D+1)}$

(3) $\mathbb{E}[L_i \mid B_i] = \mathbb{E}[L_i \mid A_i] = \mathcal{O}(D)$

(4) $\mathbb{E}[L_j \mid B_i] = \mathbb{E}[L_j \mid \neg A_j] = \mathcal{O}(1)$

Therefore,

$$
\begin{aligned}
\mathbb{E}[T] &= \sum_{i=1}^{\infty} \mathbb{E}[T \mid B_i] \cdot \Pr(B_i) \\
&= \sum_{i=1}^{\infty} \Big(\sum_{j=1}^{i-1} \mathbb{E}[L_j \mid B_i] + \mathbb{E}[L_i \mid B_i] \Big) \cdot (1 - 2^{-(D+1)})^{i-1} 2^{-(D+1)} \\
&= \sum_{i=1}^{\infty} (\mathcal{O}(i) + \mathcal{O}(D)) \cdot (1 - 2^{-(D+1)})^{i-1} 2^{-(D+1)} \\
&= 2^{-(D+1)} \cdot \sum_{i=1}^{\infty} \mathcal{O}(i) \cdot (1 - 2^{-(D+1)})^{i-1} \\
&\qquad + \mathcal{O}(D) \cdot 2^{-(D+1)} \cdot \sum_{i=1}^{\infty} (1 - 2^{-(D+1)})^{i-1} \\
&= 2^{-(D+1)} \cdot \mathcal{O}(2^{2D}) + \mathcal{O}(D) \cdot 2^{-(D+1)} \cdot 2^{D+1} = \mathcal{O}(2^D) \ . \qquad \square
\end{aligned}
$$

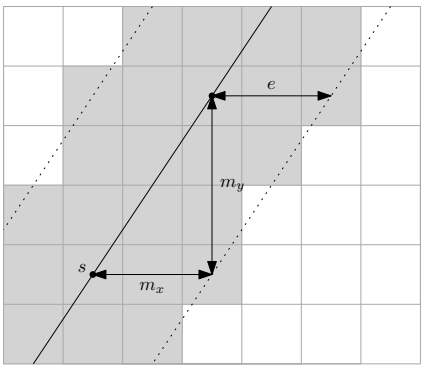


Figure 5.4: The gray cells belong to a band with slope (m_x, m_y) and extent e.

5.4 Two Agents

Our goals in this section are to show, on the negative side, that two deterministic FA-agents *cannot* solve sync-ANTS, and, on the positive side, that one deterministic FA-agent together with one deterministic PDA-agent *can* solve sync-ANTS.

5.4.1 No Deterministic FA-Protocol

We start off with proving the first result. Before doing so, we define the notion of a *band* in $\mathbb{Z}^2$. A band is the discrete version of a fat line in Euclidean space, i.e., the set of cells that have at most a certain distance from a line. See Figure 5.4 for an illustration.

Definition. A *band* $B = (s, m, e)$, $s = (x_s, y_s) \in \mathbb{Z}^2$ with slope $m = (m_x, m_y) \in \mathbb{Z}^2$ of extent $e \in \mathbb{Z}^+$ consists of all cells c for which there exists a point $p = (s_x + \lambda m_x, s_y + \lambda m_y)$ for some $\lambda \in \mathbb{R}$ such that $\|c - p\|_1 \leq e$ where $\|x\|_1$ denotes the ℓ_1-norm of x.

Observation 5.4. *Let $\mathcal{B}$ be a finite set of bands with finite extent. Then $\mathbb{Z}^2 \setminus \bigcup_{B \in \mathcal{B}} B \neq \emptyset$.*

Proof. Assume for the sake of a contradiction that the bands in $\mathcal{B}$ cover $\mathbb{Z}^2$ completely. Let e^* be the maximum extent of the bands in $\mathcal{B}$. Consider a square region S of $\mathbb{Z}^2$ with ℓ^2 cells for $\ell > 2|\mathcal{B}|e^*$ and a fixed band $B = (s, m, e) \in \mathcal{B}$. Assume wlog. that $|m_x| \leq |m_y|$. Observe that $|B \cap S| \leq \ell \cdot 2e^*$ since S vertically extends over ℓ cells and the horizontal width of $B \cap S$ is at most $2e^*$. Let $A = \bigcup_{B \in \mathcal{B}} B$ and we get $|A \cap S| \leq 2|\mathcal{B}|e^* \cdot \ell < \ell^2 = |S|$. Thus, the bands in $\mathcal{B}$ do not even cover the cells in S, a contradiction. □

We denote by $M(\mathcal{P}) = (t_i)_{i>0}$ the strictly increasing sequence of all points in time when two agents meet during the execution of protocol $\mathcal{P}$. An important ingredient for the proof is the following lemma, which holds for an arbitrary amount of agents.

Lemma 5.5. *If $\mathcal{P}$ is an effective deterministic FA-protocol for sync-ANTS, then $|M(\mathcal{P})| = \infty$.*

Proof. Assume for the sake of contradiction that $\mathcal{P}$ is an effective deterministic protocol with finite $|M(\mathcal{P})|$. Thus, there exists a largest point in time $t^* = \max(M(\mathcal{P}))$ when two agents meet and after which no two agents meet anymore and the number of cells explored until t^* is finite. Consider now agent a and let q be the state that has been entered by agent a twice after t^* at the earliest time. Let $(t_i)_{i>0}$ be the strictly increasing sequence of points in time after t^* when a enters state q and denote $I_i = [t_i, t_{i+1}]$. Observe that the behavior of a in each interval I_i is identical, hence a will keep on repeating the same transitions and movements as in I_1 forever. Observe further that a can only move a finite distance in each I_i as it has a finite length.

Consider the vector $v_i(a) = C_a(t_{i+1}) - C_a(t_i)$ describing the net-translation of a during I_i and observe that by the above argument $v_i(a) = v_1(a)$ for all $i > 0$. There are two cases: If $v_1(a) = 0$, then agent a explores only a constant amount of cells for $t \to \infty$. If $v_1(a) \neq 0$, then a exhibits a net-movement into the direction of $v_1(a)$ in each I_i and since it only explores a constant amount of cells in each I_i, agent a explores only cells in a band with finite width after t^*. By Observation 5.4, the agents cannot explore all cells in $\mathbb{Z}^2$ and the claim follows. □

Theorem 5.4. *There exists no effective deterministic FA-protocol for* sync-ANTS *for* $n = 2$.

Proof. Assume for the sake of contradiction that $\mathcal{P}$ is an effective deterministic protocol for two agents a_1 and a_2. By Lemma 5.5 we know that $|M(\mathcal{P})| = \infty$. Let Q_1 and Q_2 be the set of states of the two FAs controlling a_1 and a_2. We denote by $Q_1(t) \in Q_1$ and $Q_2(t) \in Q_2$ the state of agent a_1 and a_2 at time t and further $Q(t) = (Q_1(t), Q_2(t))$. Observe that since $|M(\mathcal{P})| = \infty$, there must be a pair of states $(q_1, q_2) \in Q_1 \times Q_2$ such that the sub-sequence $T = (\tau_i)_{i>0}$ of $M(\mathcal{P})$ that consists of all $\tau \in M(\mathcal{P})$ such that $Q(\tau) = (q_1, q_2)$, is infinite. We denote the intervals $I_i = [\tau_i, \tau_{i+1}]$ and observe that a_1 and a_2 (individually) perform exactly the same state transitions and movements in each interval I_i (agent a_1 and a_2 might meet between τ_i and τ_{i+1} in different states, but their behavior is fully determined by their states at time τ_i). Thus, there is a fixed vector $v = C_{a_1}(\tau_{i+1}) - C_{a_1}(\tau_i)$ representing the translation of the meeting cell of a_1 and a_2 during some I_i and furthermore a fixed constant $\vartheta > 0$ such that $\tau_{i+1} - \tau_i = \vartheta$. Consequently, a_1 and a_2 can only explore cells in a band with finite width after τ_1. Since $E(\tau_1)$ is finite, Observation 5.4 yields a contradiction. □

5.4.2 Deterministic FA/PDA-Protocol for sync-ANTS

The second result of this section establishes that while two agents controlled by a FA do not allow for an effective deterministic protocol for sync-ANTS, one FA-agent and one PDA-agent do so.

The protocol is essentially an adapted version of the protocol from Section 5.3.1. The Explorer behaves identically to Section 5.3.1 and performs diamond searches with increasing distances to the origin. The second PDA-agent

replaces the two Guides by walking along the axis in order to signal to the Explorer when the search in a quarter-plane is complete and it should therefore alter its movement direction. The trick here is that the Guide tracks its distance from the origin using the stack. More precisely, the Guide pushes a symbol onto the stack whenever it performs a movement outwards on one of the axes and pops one symbol from the stack whenever it moves towards the origin. Using this trick, the Guide can detect when it has arrived at the origin by verifying whether the stack is empty, i.e., the top-most symbol is ε. Then the algorithm works as follows:

At time $t = 0$, the Guide and the Explorer both move one cell north (and the Guide records this move on the stack). Whenever the two agents are located together on the north-axis in cell $(0, d)$, the Explorer starts a diagonal walk towards south-west while the Guide moves south towards the origin until it arrives there, which it can track using the stack. Upon arriving there, it moves west until it meets the Explorer. As the length of the two (different) paths from cell $(0, d)$ to cell $(-d, 0)$ is equal, both the Guide and the Explorer arrive in cell $(0, -d)$ at the same time. Now the Explorer changes its movement direction and the Guide moves back to the origin after which it moves south to meet the Explorer on the south axis in cell $(0, -d)$. They repeat this process to meet on the west axis in cell $(d, 0)$ and on the north axis in cell $(0, d)$. When the Explorer has completed the diamond search of level d by arriving at cell $(0, d)$ again, it moves together with the Guide to cell $(0, d + 1)$ and the search of level $d + 1$ begins.

It is easy to see that the above algorithm guarantees that the Explorer meets the Guide every time it crosses an axis and that therefore any level d is explored in finite time.

Theorem 5.5. *There exists an effective deterministic protocol for sync-ANTS for $n = 2$ that uses one FA-protocol and one PDA-protocol.*

5.4.3 Deterministic PDA-Protocol for async-ANTS

Since two PDAs can simulate a Turing machine [62] by using both their stacks to represent the infinite band of the Turing machine, it is not surprising that two PDAs allow for an effective deterministic protocol for async-ANTS. We show that two PDA-agents can find the treasure by only using their counting capabilities, i.e., using an alphabet containing only a single symbol.

The two agents a and b employ the following protocol: Both agents walk "hand-in-hand", i.e., have a distance of at most one at all times, and perform a spiral search with increasing distances from the origin (cf. Section 5.3.1). At any time during the execution, they maintain the invariant that the sum of the number of symbols on both stacks equals their distance from the origin. They start from the cell $(0, 1)$ with the stack of agent a containing one symbol. When the two agents start a spiral search from cell $(0, i)$, agent a has i symbols on is stack. When a and b walk south-west, agent a removes a symbol from its stack every other step while agent b pushes one symbol to its stack every other step. When the stack of agent a is empty, agent b's stack contains i symbols and the agents have arrived at the cell $(-i, 0)$ on the west axis. Then they reverse their

roles and move together to the south, east, and again north axis in the same fashion to finish the search in distance i. Thereafter, they move one cell north, push one additional symbol to the stack to account for the increased distance and start a new search in distance $i + 1$. It is easy to see that this protocol can be implemented to work in an asynchronous environment and guarantees that the two agents locate the treasure.

Theorem 5.6. *There exists an effective deterministic PDA-protocol for* async-ANTS *for* $n = 2$.

5.5 One Agent

In this section we show that neither a single randomized FA-agent nor a single deterministic PDA-agent can find the treasure in finite time while a randomized PDA-agent is able to do so.

5.5.1 No Randomized FA-Protocol

Consider a single agent who is controlled by a finite state machine. The movements the agent performs on the grid can be described by a Markov chain, where we simply assign the state set of the Markov chain to be the states of the finite state machine and the transition probabilities to be the probabilities assigned to the corresponding state transitions. Clearly, this Markov chain is finite and all state transition probabilities are constants. Therefore, it must contain an irreducible subset H of states that are entered within constant amount of state transitions with a constant probability; for the rest of the section, we condition on this event and focus on the restriction of the Markov chain to the states of H. If we show that the expected time to reach the treasure is infinite under this event, it follows that the expected time to reach the treasure is infinite in general, since this event occurs with constant probability.

Let s be the first state in H visited by the Markov chain. Let d be the distance of the agent from the origin when the Markov chain visits state s for the first time and recall that d is bounded from above by some constant. Since H is an irreducible set of states with finite cardinality $|H|$, standard Markov chain theory (see, e.g., [5]) ensures that state s is visited infinitely often with probability 1 and the expected time between any two such visits is finite.

Assume hereafter that the the component H of the Markov chain is aperiodic. This assumption is without loss of generality, since by augmenting each state of the Markov chain with a self transition that occur with a constant probability, one obtains an aperiodic Markov chain without increasing the expected hitting times of the agent by more than a constant factor.

For our purposes, it is enough to study the movements in one dimension and therefore, we project the location of the agent on the x-axis and ignore its movements in the y-dimension. So, in what follows, we focus on a single agent that traverses $\mathbb{Z}$ and let $X_i \in \mathbb{Z}$, $i = 1, 2, \ldots$, be the (x-coordinate of the) location of the agent at the i^{th} time the Markov chain visits state s.

Assume towards contradiction that the agent hits every point $x \in \mathbb{Z}$ in finite expected time. Since the Markov chain controlling the movements of the agent is irreducible and aperiodic, it follows that the agent hits every point $x \in \mathbb{Z}$ while in state s in finite expected time, namely, the random variable $T(x) = \min_{i \geq 1} X_i = x$ has finite expectation.

The stochastic process $\{X_i\}_{i \geq 1}$ is, by itself, Markovian, i.e.,

$$\Pr(X_n = x_n \mid X_1 = x_1, \ldots, X_{n-1} = x_{n-1}) = \Pr(X_n = x_n \mid X_{n-1} = x_{n-1}) \ .$$

This process has been characterized by Feller [56, p. 396, Theorem 2] based on the random variable $Y = X_{i+1} - X_i$, concluding that:

(1) if $\mathbb{E}[Y] > 0$, then X_i drifts to the right and $\mathbb{E}[T(x)] = \infty$ for any $x < -d$;

(2) if $\mathbb{E}[Y] < 0$, then X_i drifts to the left and $\mathbb{E}[T(x)] = \infty$ for any $x > d$; and

(3) if $\mathbb{E}[Y] = 0$, then X_i is null-recurrent and $\mathbb{E}[T(x)] = \infty$ for any x such that $|x| > d$.

In all three cases we reach a contradiction to the assumption that $\mathbb{E}[T(x)]$ is finite for all $x \in \mathbb{Z}$, thus establishing the following theorem.

Theorem 5.7. *There exists no effective randomized FA-protocol for* sync-ANTS *for* $n = 1$.

5.5.2 No Deterministic PDA-Protocol

Consider a single agent controlled by a *deterministic* PDA-protocol. We denote by $S(i)$ the size of the stack, i.e., the number of symbols on the stack (directly) after step i and by $C(i) = (q, \gamma)$ the tuple of the state $q \in Q$ and the top-most stack symbol $\gamma \in \Gamma$ (directly) after step i. Let $\mathcal{C} = Q \times \Gamma$ be the set of all configurations and observe that $|\mathcal{C}|$ is constant. As the behavior of a PDA is fully determined by its state and the top-most stack symbol, the following observation is immediate.

Observation 5.6. *Let* $0 < i_1 < i_2$ *be two different steps with* $C(i_1) = C(i_2)$ *and let* i_2 *be the smallest such index. If* $S(i) \geq S(i_1)$ *for all* $i_1 \leq i \leq i_2$, *then* $C(j) = C(j + k \cdot (i_2 - i_1))$ *for all* $i_1 \leq j \leq i_2$ *and* $k \in \mathbb{Z}_0^+$.

Note that the observation also implies that the agent executes the identical sequence of actions between step i_1 and i_2.

Observe that, since any protocol must be able to run for an arbitrary time, we can partition the set $\mathcal{C}$ into the configurations $\mathcal{C}_f$ containing all configurations that are entered finitely often and the configurations $\mathcal{C}_\infty$ that are entered infinitely often during the execution of a given protocol. Observe that there exists step i_∞ such that $C(i) \in \mathcal{C}_\infty$ for any step $i > i_\infty$. The following lemma essentially states that after a certain step $i_r > i_\infty$, the PDA will keep on repeating its behavior with a finite period Δ (see Figure 5.5 for an illustration).

Lemma 5.7. *There exists an index* $i_r > i_\infty$ *and a period* $\Delta \in \mathbb{Z}_0^+$ *such that for all steps* i *with* $i_r \leq i < i_r + \Delta$ *we have* $C(i + k \cdot \Delta) = C(i)$ *for all* $k \in \mathbb{Z}_0^+$.

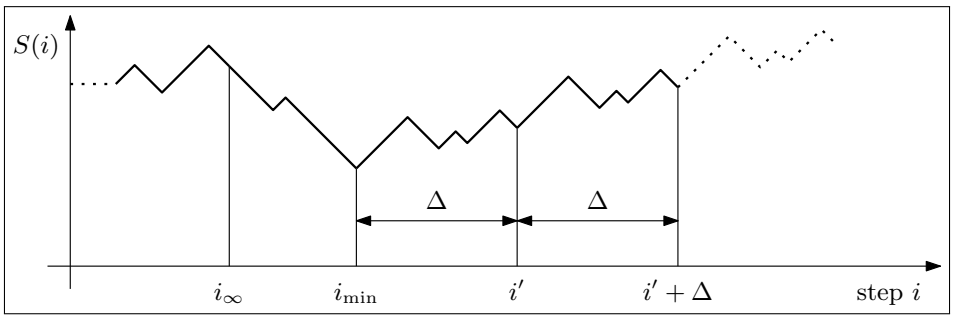


Figure 5.5: The size $S(i)$ of the stack varies for the different steps. All configurations entered after step i_∞ are entered infinitely often. The stack exhibits its minimal size after i_∞ at step $i_{\min}$ while $C(i_{\min})$ is entered again for the first time at time i'. Then the PDA will keep repeating its behavior after $i_{\min}$ with period $\Delta = i' - i_{\min}$.

Proof. Let $s_{\min} \in \mathbb{Z}_0^+$ be the minimum stack size after i_∞ and let $i_{\min}$ be the smallest index $i > i_\infty$ for which $S(i) = s_{\min}$. Let $i' > i_{\min}$ be the smallest step such that $C(i') = C(i_{\min})$. By definition of $i_{\min}$ there exists no index $i > i_{\min}$ with $S(i) < S(i_{\min})$. Thus, $i_{\min}$ and i' satisfy the preconditions of Observation 5.6 and the claim follows for $i_r = i_{\min}$ and $\Delta = i' - i_{\min}$. □

As the PDA keeps on repeating its behavior after step i_r with constant period Δ, the agent can only explore cells in a band of finite width after i_r. As i_r is finite and thus $E(i_r)$, the set of cells explored up to round i_r, is also finite, Observation 5.4 implies the following theorem.

Theorem 5.8. *There exists no effective deterministic PDA-protocol that for* sync-ANTS *for* $n = 1$.

5.5.3 Randomized PDA-Protocol for async-ANTS

The randomized protocol is an adapted version of the randomized FA-protocol for three agents from Section 5.3.2. There, one agent repeatedly performs geometric searches to a random cell in a geometrically distributed distance. It uses the two other agents to find its way back to the origin in order to start the next iteration of the search. A single agent employing a randomized PDA-protocol can do the same by using the stack to record its distance to the origin and thereby, it can perform a geometric search and then return to the origin for the next iteration. More precisely, the agent performs a geometric search as in Section 5.3.2 but whenever moving north/east/south/west, it pushes N/E/S/W, respectively, to the stack. When one geometric search ends, the agent can re-track its steps by walking north/east/south/west when reading S/W/N/E, respectively, and ends up at the origin when the stack is empty. Then, it can start the next iteration. It is easy to see that the analysis from Section 5.3.2 applies identically.

Theorem 5.9. *There exists an effective randomized PDA-protocol for* async-ANTS *for* $n = 1$.

5.6 Returning to the Origin

In this section we briefly explain the techniques through which the previously mentioned protocols can be amended in order to guarantee that, upon locating the treasure, all agents return to the origin in a timely manner, i.e., with a constant multiplicative overhead in terms of the runtime.

For all deterministic protocols, the idea is simply that upon locating the treasure, the agent(s) invert the search and progressively move closer towards the origin in the same manner as they moved further away from the origin in the search stage. More concretely, when the Explorer locates the treasure, it first finishes the search of the current distance from the origin (the current triangle or diamond in the 4-ant and 3-ant protocol, respectively) to move all guides into a well-defined position. Then it starts a walk along the triangle/diamond in the next distance but notifies the Guides that it meets on the way that the treasure has been found and that they should also retrace their steps back towards the origin. Clearly, this technique ensures that all the agents eventually end up at the origin. In the two deterministic protocols involving PDAs, returning to the origin is even simpler. After locating the treasure, the agents continue until they arrive at the next axis and then, knowing the distance to the origin, they simply walk back along the axis until they arrive at the origin.

As both randomized protocols involve that all agents repeatedly return to the origin, the agents can just follow the same procedure upon locating the treasure.

5.7 Conclusion

The variety of results of this chapter are summarized in Table 5.1. While our findings almost completely cover the landscape of problem configurations, Table 5.1 essentially shows two gaps, which, in our opinion, represent interesting open problems: Can two agents controlled by a randomized FA solve the synchronous or asynchronous version of the ANTS problem? Is there an effective FA-protocol for async-ANTS for three agents when no random bits are available? We conjecture that the answer two both questions is "no", but our efforts to prove this have not been fruitful, yet.

Problem	**FA**				**PDA**			
	sync		async		sync		async	
	det	rand	det	rand	det	rand	det	rand
One agent		$\times^{5.7}$		$\times^{5.7}$	$\times^{5.8}$	$\checkmark^{5.9}$	$\times^{5.8}$	$\checkmark^{5.9}$
Two agents	$\times^{5.4}$	?	$\times^{5.4}$	?	$\checkmark^{5.5,5.6}$		$\checkmark^{5.6}$	
Three agents	$\checkmark^{5.2}$	$\checkmark^{5.3}$	?	$\checkmark^{5.3}$				
Four agents			$\checkmark^{5.1}$					

Table 5.1: The symbol $\times$ indicates that the given combination does not allow for an effective protocol while ✓ states that there does exist an effective protocol. Empty cells follow immediately from other entries while cells marked with ? represent open problems. The numbers in the superscript refer to the theorem establishing the respective result.

Part III

The Dark Side of Collaboration

6

The Power of Collusion in Online Poker

An inherent property of distributed systems is the limited control over the individual entities. Even if there is a centralized component offering a particular service, some of the participants may exploit the distributed nature of the system in order to get a better service or to gain an unfair advantage. Such *collusions* may be highly detrimental to the entire system. What is more, if there is no verified one-to-one correspondence between the users of the system and their online identities, a single user can potentially forge a large number of online identities and use them to its advantage. Such an attack is commonly called a *Sybil attack*.[1]

In this paper, we study the feasibility and the effectiveness of a Sybil attack in a prototypical setting, online poker platforms, where collusion prevention is of utmost importance. These platforms are indeed an ideal test case as it is typically easy to create new player identities, a prerequisite for a Sybil attack. Given that there are millions of players worldwide, and there is quite a lot of money involved, it is further in the operators' best interest to circumvent any fraudulent behavior. While many operators claim that they monitor their systems for collusion attempts, little is known about the actual security mechanisms employed by these platforms.

In order to perform such an attack, we implemented a central component that interacts with a set of bots posing as regular online poker players. The bots were instructed to play at poker tables with six players in total because these tables are popular and the number of players is small enough so that the bots can

[1] The attack is named after a book about the treatment of a woman called Sybil for dissociative identity disorder.

easily control a large part of the action at the table. We focused on *No Limit Texas Hold'em*, the most popular variation of the standard card game *poker*, where each player gets two cards, and five "community cards" are shared among all the players.[2] The bots exchange information on their two hand cards with the other bots through the central component. This information is taken into account when computing the odds of winning, which in turn is a key factor when deciding whether to keep playing or to fold. Details about the implementation of the bots and their strategy is provided in Section 6.2.

The contributions of this study are the following. We show that it is indeed feasible to run a successful Sybil attack, despite what the operators of such platforms claim. While the operators state that they monitor the platform for collusion attempts, our attack was not detected to the best of our knowledge. As mentioned above, apart from investigating the feasibility of a Sybil attack, the goal is also to analyze the direct impact of collusion. We quantify the effectiveness of collusion by measuring the average gain at play money tables, when employing different numbers of bots. Our bots easily achieve a positive gain, which grows with an increasing number of bots, thereby revealing the effectiveness of our scheme. The experiments and the results are discussed in detail in Section 6.3.

6.1 Related Work

The Sybil attack [44] exploits the fact that it is hard or even impossible to prevent a user from presenting itself as multiple (online) entities in any distributed system without a trusted authority that is able to reliably associate each entity with a distinct user. This limitation allows a single user to perform a collusion attack. Collusion itself is a fundamental problem that has been studied in various fields. A large body of work on collusion exists in the area of game theory and, in particular, in economics: Researchers studied when collusion in firms is an issue depending on the internal organization [78]. Optimal lending contracts for banks, subject to potential collusion, have also been studied [77]. Collusion-proof mechanisms are further needed for *auctions* [20,28,71,92], a well studied concept in game theory. Byzantine faults [86] in distributed systems are related to collusion as the worst-case situations typically involve collusive behavior among the Byzantine entities.

As there is typically no trusted authority in peer-to-peer networks, they are vulnerable to Sybil and collusion attacks, and a lot of research has been conducted on how to mitigate this problem. One approach is to combine subjective reputations with short-term histories for verification [55]. Alternatively, clustering can be used together with quorum-based operations to minimize the effect of multiple fake entities. This approach has to be combined with a random insertion algorithm to reduce the risk of having many fake entities in the same cluster [7]. An interesting question is whether collusion actually occurs in such systems. An

[2] A player's hand consists of the best five cards out of his two private and the five community cards. See http://en.wikipedia.org/wiki/Texas_hold_'em for a detailed explanation of the rules.

empirical study on the prevalence of collusion in Maze, a peer-to-peer file-sharing system, has been carried out. Substantial evidence for collusion-like behavior has indeed been found by analyzing traffic logs [110].

The game of poker has also been studied extensively. In one line of research, the goal is to create a robot that plays competitively against human players, similar to the research on chess. However, most research focuses on *low-limit* poker in which the betting amounts adhere to a restricted format in order to significantly reduce the complexity of the decision making process. Gilpin and Sandholm introduced a new way to compute an abstraction of the game tree for heads-up, i.e., two-player, low-limit Texas Hold'em [60]. Another approach is to base game decisions on opponent modeling. Billings et al. found that robots based on opponent modeling are "reasonably strong" when playing online against human opponents at play money tables [24, 25]. An alternative strategy is to consider recorded scenarios similar to the current game state and to make a decision based on past outcomes [109]. Finally, strategies that converge to an ϵ-Nash equilibrium have been proposed [91].

Surprisingly, there is little work on collusion attacks in online poker. It has been shown that collusion could be detected in many cases by analyzing the game data and modeling the player behavior [103]. The work most related to ours is due to Simm who built a colluding poker robot for no limit Texas Hold'em [102]. His tests revealed that collusion can provide an advantage when playing against other robots, i.e., the effectiveness against human opponents was not studied. To the best of our knowledge, there is no work on the effectiveness of collusion on online poker platforms. In particular, our setup in which (simple) robots collude to gain an advantage over human players has not been considered before.

6.2 Colluding Bots

We will now discuss the technical details of our attack. Apart from describing the architecture, the interaction between the components, and the bots' poker strategy, we will also present the identified collusion detection mechanisms that are used by the poker platform and our techniques to overcome them.

6.2.1 Preliminaries

Our initial task was to identify a selection of online poker platforms suitable for our purposes. We based the selection process on the following criteria:

1. We require the existence of a web-based game client since this makes it much harder for the platform providers to detect our bot. Game clients that run natively on computers have full access to the list of running processes and other data, which enables them to detect our bot more easily.

2. Preferably, the bots can directly select a certain poker table. It is harder to perform a Sybil attack when platforms automatically assign a table to their players because this requires the bots to change tables until they find each other.

3. The platform should have a sufficiently large user base in order to guarantee that there are enough active tables at any point in time.

4. In order to test the strength of the security measures used in practice, we specifically target platforms that claim to actively protect themselves against cheating/colluding players and/or bots.

It is worth noting that since the majority of the large online poker platforms share a common user base and just provide different clients or welcome bonuses, almost all of them meet the third requirement. We managed to find a suitable state-of-the-art platform that satisfies the other three criteria as well; however, we refrain from disclosing its identity and simply refer to it as "the poker platform" in the remainder of this paper in order not to tarnish the platform's reputation. It would further be unfair to state the name of the platform that we considered because we presume that the attack is equally effective on other platforms as well.

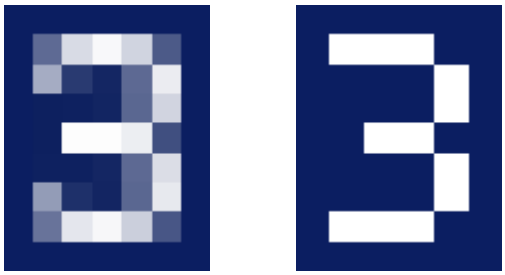
The poker strategy employed by our bots is deliberately kept simple in order to be able to gauge the direct effect of collusion.

6.2.2 Architecture

This section gives a brief overview of the architecture behind our attack.

User Interface Interaction. Since no platform provides a programming interface (API) to interact with the game client for obvious reasons, we had to solve two basic problems: First, we need to acquire information from the game client (available tables, current game situation, current bets, play options, community cards, etc.), and second, we have to send instructions to the client (select a certain table, bet a certain amount, fold, leave the table, etc.). We solve the former problem by repeatedly analyzing screenshots of the game client with a combination of optical character recognition and pattern matching with previously acquired screenshots of sub-elements (buttons, displayed cards, etc.). This allows us to create a logical representation of the important aspects of the current game state, which can then be used by the bot for further processing. Sending instructions to the game client is accomplished by simulating user interactions such as clicking and typing using a third-party library. In order to foil potential attempts by the poker platform to detect our bots, we introduced some randomness to the interactions (delays, click positions, bet sizes, etc.), which renders the bots' behavior more human-like.

Client-Server Model. Our bots communicate with each other through a server, i.e., the server acts as the central component for information exchange. When our attack is launched, all available bots start the web client, log into the poker platform, and register with the server. Subsequently, the server coordinates the collusion by instructing already registered bots which tables they should join. As there is no need for direct communication between the bots, they send all relevant data to the server from where it is distributed.



(a) Anti-Aliasing (b) No Anti-Aliasing

Figure 6.1: Character recognition/pattern matching is a challenging problem since the same character might occur in different positions with varying formatting, colors, and levels of post-processing.

6.2.3 Poker Strategies

As mentioned before, the main goal is to show the benefit of the collusion between the individual bots and *not* to develop a sophisticated poker bot that is able to play competitively against human opponents by itself. Naturally, we still require a basic strategy to be able to play.

At the core of our strategy lies the odds calculation. Since the exact computation would take too long due to the sheer number of possible hands, we used the Monte Carlo method to approximate the winning odds by uniformly sampling games and computing how often the bot's hand would win.

The strategy is based on the following straightforward rule: A bot only stays in the game if the expected win is larger than the expected loss. More formally, if w is the amount it can win, c is the amount it must invest to stay in the game, and p is the computed odds of winning, the ratio

$$\rho := \frac{wp}{c(1-p)}$$

must be greater than 1. Empirical tests revealed that other thresholds for ρ are more appropriate depending on the game state. In total, we used four different variants of this strategy using different parameters when to fold, call, or bet. Since all variants are fundamentally the same, and due to lack of space, we dispense with a more thorough discussion.

6.2.4 Collusion

Our basic collusion scheme is simple. The bots playing at the same table exchange information about their hand cards at the start of each round when the hand cards are distributed to all players. This information is then used in the odds calculation algorithm since all of these cards cannot appear as community cards or hand cards of the other opponents, i.e., the bots already benefit from the collusion at this stage.

In some scenarios, it is not easy to decide how many bots should stay in the game, especially when no community cards have been revealed yet, i.e., before

the flop. In order to keep the strategy simple, we allow the bots to bet against each other and to play independently using the poker strategy described in the previous section only before the flop. After the flop, the bots decide collectively which hand is the best with respect to the winning probability, and only the bot with the best hand will raise or call to stay in the game. All other bots will fold as soon as checking is not an option anymore.

6.2.5 Collusion Countermeasures

The poker platform uses several measures to prevent collusion among their players. First of all, it is not possible to establish two connections with the same IP address. This limitation makes it harder to implement collusion schemes but, as in our case, it is no obstacle for an attacker that has many addresses at his disposal. We circumvented this constraint by using a separate virtual machine with an individual IP address for each bot. A second, more severe restriction targets the monetary transfer system. The poker platform requires all users who wish to play at real money tables to provide identity information. More precisely, it is possible to open up an account with fake information but an identity check (passport scan) is required in order to transfer money to the account. As we did not intend to deploy our bots at real money tables, this obstacle did not affect us.

Besides those obvious methods to impede collusion, we assume that the poker platform also applies some hidden functionality, which may include the profiling of player actions, e.g., to detect behavior that is "too regular" for a human player. This assumption is reasonable as many operators state that anti-cheating mechanisms are running on their platforms. However, the effectiveness of these mechanisms seems to be limited, given that our attack remained undetected — at least to the best of our knowledge.

6.3 Evaluation

In this section we will present an evaluation of the experimental results of our attack. We use the term *game rounds* when we talk about a number of rounds played at a certain table independent of the number of bots at this table. Since we are interested in the average gain per round and per bot, we introduce the concept of a *bot round*, which refers to the number of game rounds multiplied with the number of bots at the table. For example, if four bots play 50 game rounds at the same table, then this amounts to a total of 200 bot rounds. Thus, the average gain per bot round is equivalent to the gain per game round and per bot.

Our Sybil attack was launched only at play money tables and ran for roughly one month. The bots played about 6,000 bot rounds in total.

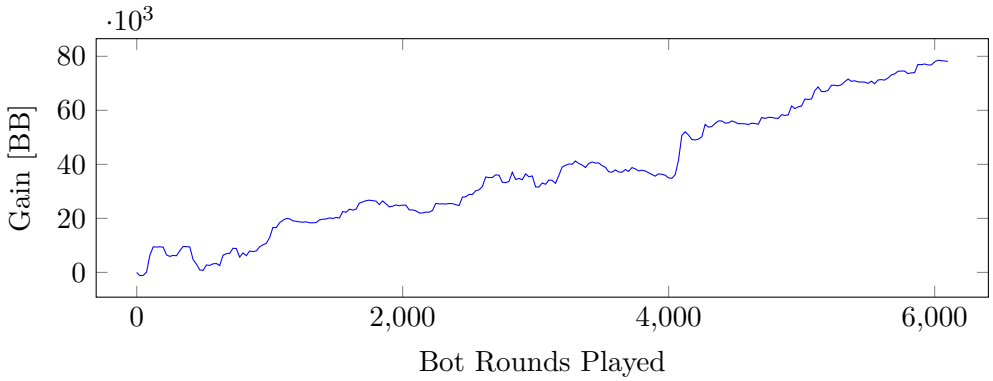


Figure 6.2: The overall gain at *play money tables* shows a trend of winning a little over one big blind (BB) per bot round.

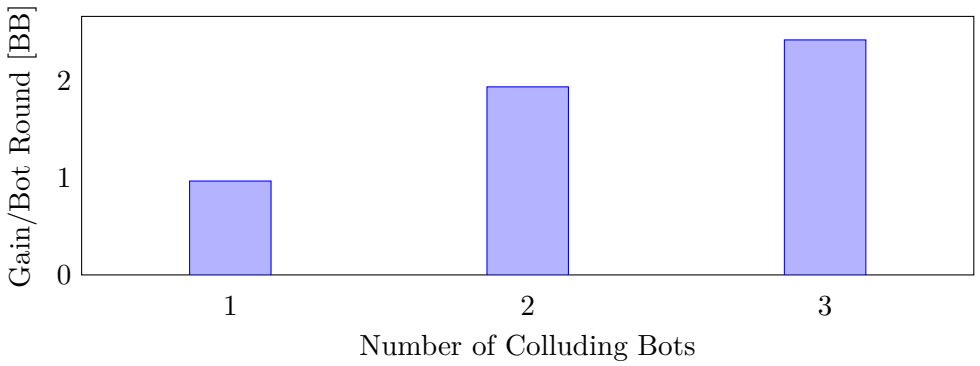


Figure 6.3: The average gain per bot round at *play money tables* increases significantly with the number of colluding bots. The charts for 1, 2, and 3 colluding bots show the average gain over 4,178, 1,201, and 690 bot rounds, respectively.

6.3.1 Results

The overall gain of our bots on play money tables is depicted in Figure 6.2. On average, the bots won slightly more than one big blind (BB) per bot round. An explanation for this impressive performance of our bots is most certainly the careless play at play money tables. This is due to the fact that every play money account can be reset to 1,000 units at any time, which both encourages a risky playing style and continuously increases the amount of play money in the system.

More importantly, the impact of increasing the number of colluding bots at the same table is shown in Figure 6.3. The chart clearly shows the benefit of collusion: Three bots achieve a gain of more than two big blinds per bot, i.e., the gain is more than twice as high per bot compared to having a single bot at the table.

6.4 Conclusion

The key finding in our study is that collusion among bots on an online poker platform is feasible and provides a significant advantage. In particular, our results show that collusive bots achieve a sizable return on investment over longer periods of time at play money tables even with a limited game intelligence, and that adding more bots results in a linear increase in the average gain per bot.

When attempting to generalize our results to real money tables, we first have to observe that real money tables naturally attract players with much higher skill levels, which engage in a significantly more serious play style. Hence, we expect the individual performance of our bots to be significantly worse than on play money tables. Implementing a smarter game strategy that goes beyond the basic approach of basing all decisions solely on the expected win, e.g., by incorporating techniques such as opponent modeling and profiling, (semi-)bluffing and so on, should help to alleviate this short-coming.

Concerning our results on the effect of collusion between multiple bots, we conjecture, that similar results should be observable also at real money tables as knowing more about the cards in the deck can only help one's game. We did not attempt to verify this conjecture because of the obvious ethical pitfalls, but our findings on play money tables are promising.

In light of these findings, it becomes an even more pressing need for poker platform providers to come up with and deploy effective countermeasures to protect their system from such collusion attacks. While our attack was not detected, despite its simplicity and the little effort expended to conceal it, we believe that it is possible to detect such attacks by monitoring the game state and analyzing log data. In case of a collusion, the game logs will likely reveal certain patterns that are not observed in regular play, such as groups of players that often play together but rarely bet against each other. Creating a separate profile for each player is another viable starting point for a detection mechanism. Apart from collusion detection, some more effort should also be invested into collusion prevention. A simple method is to tighten the control over the system, e.g., by placing players at randomly chosen tables, and disallowing frequent table chances. While this step raises the bar for a successful Sybil attack, it comes at the expense of the players' freedom to play (together) at any table of their choosing.

Part IV

Conclusions & References

7

Conclusions

At the beginning of this work, we set of with the goal to advance our current understanding of various aspects of collaboration in distributed systems. We have shown how collaboration can be incentivized in matching markets. In a stable matching, partners are less likely to break up with each other in order to find a better match as no mutually improving pairing exists. We then turned towards mobile robots/agents and showed how they can work together to quickly gather at a point in the plane and how a whole colony of ants or just a few individual ants can collaboratively find a food source efficiently. Lastly, we considered how one can abuse collaboration in the form of collusion to gain an unfair advantage in online poker. These examples constitute a diverse selection of the importance and the power of collaboration, whether in nature, technology, games, or in our own lives.

Another area, in which the significance of collaboration cannot be emphasized enough is the kind of work that resulted in this thesis: research. All over the world, thousands of young researchers are tilting against the windmills of academia as PhD students or PostDocs. In all the competition for recognition in the form of accepted papers at top conferences and journals or funding from research grants, they often tend to forget that the main purpose of science is not the progress of the individual but rather the advancement of knowledge per se. Even more so, it is commonly accepted that collaboration among scientists increases the efficiency of the involved individuals. Top-tier journals such as *Science* or *Nature* show a positive correlation between the number of authors of a publication and its impact and more than half of the Nobel Prizes in Physics, for example, have been awarded to a group of people for their ground-breaking collaborations. A recent study by Bahrami et al. [22] showed that, in general, the outcome of a task improves when two people work together compared to

their individual performance.

As a personal example from my PhD studies, I recall working alone on an early version of the ANTS problem (covered in Chapter 4 and 5) and having difficulties in how to approach our variant of the problem formally. After fiddling around with the problem for quite some time without significant progress, I turned to my fellow PhD student Jara Uitto and within barely two weeks, we had drawn up the essence of the article that served as foundation for Chapter 4.

My hope is that scientists — and in particular young researchers — occasionally remember the classic and pure character of science, in which collaboration plays a central role. To conclude this work, I like to quote Isaac Asimov who wittily summarizes my previous thoughts as follows.

> *"A knotty puzzle may hold a scientist up for a century, when it may be that a colleague has the solution already and is not even aware of the puzzle that it might solve."*
>
> — Isaac Asimov, The Robots of Dawn

Bibliography

[1] Agmon, N., Peleg, D.: Fault-tolerant gathering algorithms for autonomous mobile robots. In: Proceedings of the 15th ACM-SIAM Symposium on Discrete Algorithms (SODA). (2004) 1070–1078

[2] Aigner, M., Fromme, M.: A game of cops and robbers. Discrete Applied Mathematics **8** (1984) 1–12

[3] Albers, S., Eilts, S., Even-Dar, E., Mansour, Y., Roditty, L.: On nash equilibria for a network creation game. In: Proceedings of the 17th ACM-SIAM Symposium on Discrete Algorithms (SODA). (2006) 89–98

[4] Albers, S., Henzinger, M.: Exploring unknown environments. SIAM Journal on Computing (SICOMP) **29** (2000) 1164–1188

[5] Aldous, D., Fill, J.A.: Reversible Markov chains and random walks on graphs (2002) Unfinished monograph, recompiled 2014, available at `http://www.stat.berkeley.edu/~aldous/RWG/book.html`.

[6] Aleliunas, R., Karp, R.M., Lipton, R.J., Lovasz, L., Rackoff, C.: Random walks, universal traversal sequences, and the complexity of maze problems. In: Proceedings of the 20th Annual Symposium on Foundations of Computer Science (SFCS). (1979) 218–223

[7] Anceaume, E., Ludinard, R., Ravoaja, A., Brasileiro, F.: Peercube: A hypercube-based P2P overlay robust against collusion and churn. In: Proceedings of the 2nd IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO). (2008) 15–24

[8] Andelman, N., Feldman, M., Mansour, Y.: Strong price of anarchy. In: Proceedings of the 18th ACM-SIAM Symposium on Discrete Algorithms (SODA). (2007) 189–198

[9] Ando, H., Oasa, Y., Suzuki, I., Yamashita, M.: Distributed memoryless point convergence algorithm for mobile robots with limited visibility. IEEE Transactions on Robotics and Automation **15**(5) (1999) 818–828

[10] Ando, H., Suzuki, Y., Yamashita, M.: Formation and agreement problems for synchronous mobile robots with limited visibility. In: Proceedings of the 1995 IEEE International Symposium on Intelligent Control (ISIC). (1995) 453–460

[11] Angluin, D., Aspnes, J., Diamadi, Z., Fischer, M.J., Peralta, R.: Computation in networks of passively mobile finite-state sensors. Distributed Computing (2006) 235–253

[12] Anshelevich, E., Dasgupta, A., Kleinberg, J.M., Tardos, É., Wexler, T., Roughgarden, T.: The price of stability for network design with fair cost allocation. SIAM Journal on Computing (SICOMP) **38**(4) (2008) 1602–1623

[13] Anshelevich, E., Dasgupta, A., Tardos, E., Wexler, T.: Near-optimal network design with selfish agents. In: Proceedings of the 35$^{\text{th}}$ ACM Symposium on Theory of Computing (STOC). (2003) 511–520

[14] Arkin, E.M., Bae, S.W., Efrat, A., Okamoto, K., Mitchell, J.S.B., Polishchuk, V.: Geometric stable roommates. Information Processing Letters **109**(4) (2009) 219–224

[15] Aspnes, J., Ruppert, E.: An introduction to population protocols. In Garbinato, B., Miranda, H., Rodrigues, L., eds.: Middleware for Network Eccentric and Mobile Applications. (2009) 97–120

[16] Augugliaro, F., Lupashin, S., Hamer, M., Male, C., Hehn, M., Mueller, M., Willmann, J., Gramazio, F., Kohler, M., D'Andrea, R.: The flight assembled architecture installation: Cooperative construction with flying machines. IEEE Control Systems **34**(4) (2014) 46–64

[17] Augugliaro, F., Schoellig, A., D'Andrea, R.: Dance of the flying machines: Methods for designing and executing an aerial dance choreography. IEEE Robotics Automation Magazine **20**(4) (2013) 96–104

[18] Awerbuch, B., Azar, Y., Epstein, A.: The price of routing unsplittable flow. In: Proceedings of the 37$^{\text{th}}$ ACM Symposium on Theory of Computing (STOC). (2005) 57–66

[19] Awerbuch, B., Azar, Y., Richter, Y., Tsur, D.: Tradeoffs in worst-case equilibria. Theoretical Computer Science (TCS) **361**(2) (2006) 200–209

[20] Bachrach, Y., Zadimoghaddam, M., Key, P.: A cooperative approach to collusion in auctions. ACM SIGecom Exchanges **10**(1) (2011) 17–22

[21] Baeza-Yates, R.A., Culberson, J.C., Rawlins, G.J.E.: Searching in the plane. Information and Computation **106** (1993) 234–252

[22] Bahrami, B., Olsen, K., Latham, P.E., Roepstorff, A., Rees, G., Frith, C.D.: Optimally interacting minds. Science **329**(5995) (2010) 1081–1085

[23] Bajaj, C.: The algebraic degree of geometric optimization problems. Discrete & Computational Geometry **3** (1988) 177–191

[24] Billings, D., Davidson, A., Schaeffer, J., Szafron, D.: The challenge of poker. Artificial Intelligence **134**(1-2) (2001) 201–240

[25] Billings, D., Peña, L., Schaeffer, J., Szafron, D.: Using probabilistic knowledge and simulation to play poker. In: Proceedings of the 16$^{\text{th}}$ National Conference on Artificial Intelligence and 11$^{\text{th}}$ Innovative Applications of Artificial Intelligence (AAAI/IAAI). (1999) 697–703

[26] Chatzigiannakis, I., Markou, M., Nikoletseas, S.: Distributed circle formation for anonymous oblivious robots. In: Proceedings of the 3$^{\text{rd}}$ Workshop on Efficient and Experimental Algorithms (WEA). (2004) 159–174

[27] Chazelle, B.: Natural algorithms. In: Proceedings of the 20$^{\text{th}}$ ACM-SIAM Symposium on Discrete Algorithms (SODA). (2009) 422–431

[28] Che, Y., Kim, J.: Optimal collusion-proof auctions. Journal of Economic Theory **144**(2) (2009) 565–603

[29] Chen, H.L., Roughgarden, T.: Network design with weighted players. In: Proceedings of the 18$^{\text{th}}$ ACM Symposium on Parallelism in Algorithms and Architectures (SPAA). (2006) 29–38

[30] Christodoulou, G., Koutsoupias, E.: On the price of anarchy and stability of correlated equilibria of linear congestion games. In: Proceedings of the 13$^{\text{th}}$ European Symposium on Algorithms (ESA). (2005) 59–70

[31] Christodoulou, G., Koutsoupias, E.: The price of anarchy of finite congestion games. In: Proceedings of the 37$^{\text{th}}$ ACM Symposium on Theory of Computing (STOC). (2005) 67–73

[32] Chrystal, G.: On the problem to construct the minimum circle enclosing n given points in a plane. In: Proceedings of the Edinburgh Mathematical Society, Third Meeting. (1885) 30–35

[33] Cieliebak, M., Flocchini, P., Prencipe, G., Santoro, N.: Solving the robots gathering problem. In: Proceedings of the 30$^{\text{th}}$ International Colloquium on Automata, Languages and Programming (ICALP). (2003) 1181–1196

[34] Cohen, R., Peleg, D.: Robot convergence via center-of-gravity algorithms. In: Proceedings of the 11$^{\text{th}}$ International Colloquium on Structural Information and Communication Complexity (SIROCCO). Volume 3104 of Lecture Notes in Computer Science. (2004) 79–88

[35] Cohen, R., Peleg, D.: Convergence properties of the gravitational algorithm in asynchronous robot systems. SIAM Journal on Computing (SICOMP) **34**(6) (2005) 1516–1528

[36] Czumaj, A., Vöcking, B.: Tight bounds for worst-case equilibria. In: Proceedings of the 13$^{\text{th}}$ ACM-SIAM Symposium on Discrete Algorithms (SODA). (2002) 413–420

[37] Défago, X., Konagaya, A.: Circle formation for oblivious anonymous mobile robots with no common sense of orientation. In: Proceedings of

the 2nd ACM International Workshop on Principles of Mobile Computing (POMC). (2002) 97–104

[38] Degener, B., Kempkes, B., Kling, P., Meyer auf der Heide, F.: A continuous, local strategy for constructing a short chain of mobile robots. In: Proceedings of the 17th International Colloquium on Structural Information and Communication Complexity (SIROCCO). (2010) 168–182

[39] Degener, B., Kempkes, B., Langner, T., auf der Heide, F.M., Pietrzyk, P., Wattenhofer, R.: A tight runtime bound for synchronous gathering of autonomous robots with limited visibility. In: Proceedings of the 23rd ACM Symposium on Parallelism in Algorithms and Architectures (SPAA). (2011)

[40] Degener, B., Kempkes, B., Meyer auf der Heide, F.: A local $O(n^2)$ gathering algorithm. In: Proceedings of the 22nd ACM Symposium on Parallelism in Algorithms and Architectures (SPAA). (2010) 217–223

[41] Deng, X., Papadimitriou, C.: Exploring an unknown graph. Journal of Graph Theory **32** (1999) 265–297

[42] Dieudonné, Y., Petit, F.: Self-stabilizing deterministic gathering. In: Proceedings of the 5th International Workshop on Algorithmic Aspects of Wireless Sensor Networks (ALGOSENSORS). (2009) 230–241

[43] Diks, K., Fraigniaud, P., Kranakis, E., Pelc, A.: Tree exploration with little memory. Journal of Algorithms **51** (2004) 38–63

[44] Douceur, J.: The sybil attack. In: Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS). (2002) 251–260

[45] Dynia, M., Kutyłowski, J., Lorek, P., Meyer auf der Heide, F.: Maintaining communication between an explorer and a base station. In: Proceedings of the 1st IFIP International Conference on Biologically Inspired Collaborative Computing (BICC). (2006) 137–146

[46] Dynia, M., Kutyłowski, J., Meyer auf der Heide, F., Schrieb, J.: Local strategies for maintaining a chain of relay stations between an explorer and a base station. In: Proceedings of the 19th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA). (2007) 260–269

[47] Edmonds, J.: Paths, trees, and flowers. Canadian Journal of Mathematics **17** (1965) 449–467

[48] Edmonds, J.: Maximum matching and a polyhedron with $0,1$ vertices. Journal of Research of the National Bureau of Standards **69 B** (1965) 125–130

[49] Emek, Y., Langner, T., Stolz, D., Uitto, J., Wattenhofer, R.: How many ants does it take to find the food? In: Proceedings of the 21th International

Colloquium on Structural Information and Communication Complexity (SIROCCO). (2014)

[50] Emek, Y., Langner, T., Uitto, J., Wattenhofer, R.: Solving the ANTS problem with asynchronous finite state machines. In: Proceedings of the 41st International Colloquium on Automata, Languages and Programming (ICALP). (2014)

[51] Emek, Y., Wattenhofer, R.: Stone age distributed computing. In: Proceedings of the 32nd Annual ACM Symposium on Principles of Distributed Systems (PODC). (2013)

[52] Feder, T.: A new fixed point approach for stable networks and stable marriages. Journal of Computer and System Sciences **45** (1992) 233–284

[53] Feinerman, O., Korman, A.: Memory lower bounds for randomized collaborative search and implications for biology. In: Proceedings of the 26th International Symposium on Distributed Computing (DISC). (2012) 61–75

[54] Feinerman, O., Korman, A., Lotker, Z., Sereni, J.S.: Collaborative search on the plane without communication. In: Proceedings of the 31st Annual ACM Symposium on Principles of Distributed Systems (PODC). PODC '12 (2012) 77–86

[55] Feldman, M., Lai, K., Stoica, I., Chuang, J.: Robust incentive techniques for peer-to-peer networks. In: Proceedings of the 5th ACM Conference on Electronic Commerce (EC). (2004) 102–111

[56] Feller, W.: An Introduction to Probability Theory and its Applications. (1971)

[57] Flocchini, P., Prencipe, G., Santoro, N., Widmayer, P.: Gathering of asynchronous robots with limited visibility. Theoretical Computer Science (TCS) **337**(1–3) (2005) 147–168

[58] Fraigniaud, P., Ilcinkas, D., Peer, G., Pelc, A., Peleg, D.: Graph exploration by a finite automaton. Theoretical Computer Science (TCS) **345**(2-3) (2005) 331–344

[59] Gale, D., Shapley, L.S.: College admissions and the stability of marriage. The American Mathematical Monthly **69**(1) (1962) 9–14

[60] Gilpin, A., Sandholm, T.: Better automated abstraction techniques for imperfect information games, with application to texas hold'em poker. In: Proceedings of the 6th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS). (2007)

[61] Gusfield, D., Irving, R.W.: The stable marriage problem: structure and algorithms. (1989)

[62] Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation. (1979)

[63] Irving, R.W.: Stable marriage and indifference. Discrete Applied Mathematics **48**(3) (1994) 261–272

[64] Irving, R.W., Leather, P., Gusfield, D.: An efficient algorithm for the "optimal" stable marriage. Journal of the ACM (JACM) **34**(3) (1987) 532–543

[65] Irving, R.W., Manlove, D.F., Scott, S.: The stable marriage problem with master preference lists. Discrete Applied Mathematics **156** (2008) 2959–2977

[66] Izumi, T., Izumi, T., Kamei, S., Ooshita, F.: Randomized gathering of mobile robots with local-multiplicity detection. In: Proceedings of the 11$^{\text{th}}$ International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS). (2009) 384–398

[67] Izumi, T., Katayama, Y., Inuzuka, N., Wada, K.: Gathering autonomous mobile robots with dynamic compasses: An optimal result. In: Proceedings of the 21$^{\text{st}}$ International Symposium on Distributed Computing (DISC). (2007) 298–312

[68] Johari, R., Tsitsiklis, J.N.: Efficiency loss in a network resource allocation game. Mathematics of Operations Research **29**(3) (2004) 407–435

[69] Kim, D.H., Kim, J.H.: A real-time limit-cycle navigation method for fast mobile robots and its application to robot soccer. Robotics and Autonomous Systems **42**(1) (2003) 17–30

[70] Kim, J.H., Shim, H.S., Kim, H.S., Jung, M.J., Choi, I.H., Kim, J.O.: A cooperative multi-agent system and its real time application to robot soccer. In: Proceedings of the IEEE International Conference on Robotics and Automation. (1997) 638–643

[71] Klemperer, P.: What really matters in auction design. The Journal of Economic Perspectives **16**(1) (2002) 169–189

[72] Kling, P., Meyer auf der Heide, F.: Convergence of local communication chain strategies via linear transformations: Or how to trade locality for speed. In: Proceedings of the 23$^{\text{rd}}$ ACM Symposium on Parallelism in Algorithms and Architectures (SPAA). SPAA '11 (2011) 159–166

[73] Knuth, D.E.: Marriages stables et leurs relations avec d'autres problèmes combinatoires. (1976)

[74] Koutsoupias, E., Mavronicolas, M., Spirakis, P.G.: Approximate equilibria and ball fusion. Theory of Computing Systems (2003) 683–693

[75] Koutsoupias, E., Papadimitriou, C.: Worst-case equilibria. Computer Science Review **3**(2) (2009) 65–69

[76] Kutylowski, J., Meyer auf der Heide, F.: Optimal strategies for maintaining a chain of relays between an explorer and a base camp. Theoretical Computer Science (TCS) **410**(36) (2009) 3391–3405

[77] Laffont, J.: Collusion and group lending with adverse selection. Journal of Development Economics **70**(2) (2003) 329–348

[78] Laffont, J., Martimort, D.: Collusion and delegation. The RAND Journal of Economics (1998) 280–305

[79] Lenzen, C., Lynch, N., Newport, C., Radeva, T.: Trade-offs between selection complexity and performance when searching the plane without communication. In: Proceedings of the 33rd Annual ACM Symposium on Principles of Distributed Systems (PODC). (2014)

[80] López-Ortiz, A., Sweet, G.: Parallel searching on a lattice. In: Proceedings of the 13th Canadian Conference on Computational Geometry (CCCG). (2001) 125–128

[81] Meyer auf der Heide, F., Schneider, B.: Local strategies for connecting stations by small robotic networks. In: Proceedings of the 2nd IFIP International Conference on Biologically Inspired Collaborative Computing (BICC). (2008) 95–104

[82] Ng, C., Hirschberg, D.S.: Three-dimensional stable matching problems. SIAM Journal on Discrete Mathematics (SIDMA) **4** (1991) 245–252

[83] Nisan, N., Ronen, A.: Algorithmic mechanism design. Games and Economic Behavior **35**(1-2) (2001) 166–196

[84] Panaite, P., Pelc, A.: Exploring unknown undirected graphs. In: Proceedings of the 9th ACM-SIAM Symposium on Discrete Algorithms (SODA). (1998) 316–322

[85] Papadimitriou, C.: Algorithms, games, and the internet. In: Proceedings of the 33rd ACM Symposium on Theory of Computing (STOC). (2001) 749–753

[86] Pease, M., Shostak, R., Lamport, L.: Reaching agreement in the presence of faults. Journal of the ACM (JACM) **27**(2) (1980) 228–234

[87] Prabhakar, B., Dektar, K.N., Gordon, D.M.: The regulation of ant colony foraging activity without spatial information. PLoS Computational Biology **8**(8) (2012)

[88] Reingold, E.M., Tarjan, R.E.: On a greedy heuristic for complete matching. SIAM Journal on Computing (SICOMP) **10** (1981) 676–681

[89] Reingold, O.: Undirected connectivity in log-space. Journal of the ACM (JACM) **55** (2008) 17:1–17:24

[90] Riedmiller, M., Gabel, T., Hafner, R., Lange, S.: Reinforcement learning for robot soccer. Autonomous Robots **27**(1) (2009) 55–73

[91] Risk, N.A., Szafron, D.: Using counterfactual regret minimization to create competitive multiplayer poker agents. In: Proceedings of the 9$^{\text{th}}$ International Conference on Autonomous Agents and Multiagent Systems (AAMAS). (2010) 159–166

[92] Robinson, M.: Collusion and the choice of auction. The RAND Journal of Economics (1985) 141–145

[93] Romanishin, J., Gilpin, K., Rus, D.: M-blocks: Momentum-driven, magnetic modular robots. In: Proceedings of the 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). (2013) 4288–4295

[94] Ros, R., Arcos, J.L., de Mantaras, R.L., Veloso, M.: A case-based approach for coordinated action selection in robot soccer. Artificial Intelligence **173**(9–10) (2009) 1014–1039

[95] Roth, A.E., Sotomayor, M.A.O.: Two-sided matching: a study in game-theoretic modeling and analysis. (1990)

[96] Roughgarden, T.: Potential functions and the inefficiency of equilibria. Proceedings of the International Congress of Mathematicians (ICM) **3** (2006) 1071–1094

[97] Roughgarden, T.: The price of anarchy is independent of the network topology. Journal of Computer and System Sciences **67**(2) (2003) 341–364

[98] Roughgarden, T., Tardos, E.: How bad is selfish routing? Journal of the ACM (JACM) **49**(2) (2002) 236–259

[99] Rubenstein, M., Cornejo, A., Nagpal, R.: Programmable self-assembly in a thousand-robot swarm. Science **345**(6198) (2014) 795–799

[100] Sasaki, T., Pratt, S.C.: Groups have a larger cognitive capacity than individuals. Current Biology **22**(19) (2012) 827–829

[101] Schulz, A.S., Stier Moses, N.E.: On the performance of user equilibria in traffic networks. In: Proceedings of the 3$^{\text{rd}}$ ACM-SIAM Symposium on Discrete Algorithms (SODA). (2003) 86–87

[102] Simm, J.: AI system for online poker. Master's thesis, Tallinn University of Technology (2007)

[103] Smed, J., Knuutila, T., Hakonen, H.: Can we prevent collusion in multiplayer online games? In: Proceedings of the 9$^{\text{th}}$ Scandinavian Conference on Artificial Intelligence (SCAI). (2006) 168–175

[104] Souissi, S., Défago, X., Yamashita, M.: Gathering asynchronous mobile robots with inaccurate compasses. In: Proceedings of the 25$^{\text{th}}$ Annual ACM Symposium on Principles of Distributed Systems (PODC). (2006) 333–349

[105] Suri, S., Tóth, C.D., Zhou, Y.: Selfish load balancing and atomic congestion games. Algorithmica **47**(1) (2007) 79–96

[106] Suzuki, I., Yamashita, M.: Distributed anonymous mobile robots: Formation of geometric patterns. SIAM Journal on Computing (SICOMP) **28**(4) (1999) 1347–1363

[107] Suzuki, I., Yamashita, M.: Formation and agreement problems for anonymous mobile robots. In: Proceedings of the 31$^{\text{st}}$ Annual Allerton Conference on Communication, Control, and Computing. (1993) 93–102

[108] Vetta, A.: Nash equilibria in competitive societies, with applications to facility location, traffic routing and auctions. In: Proceedings of the 43$^{\text{rd}}$ IEEE Symposium on Foundations of Computer Science (FOCS). (2002) 416–

[109] Watson, I., Rubin, J.: CASPER: A case-based poker-bot. In: Proceedings of the 21$^{\text{st}}$ Australasian Joint Conference on Artificial Intelligence: Advances in Artificial Intelligence (AI). (2008) 594–600

[110] Yang, M., Zhang, Z., Li, X., Dai, Y.: An empirical study of free-riding behavior in the maze P2P file-sharing system. Peer-to-Peer Systems IV (2005) 182–192

[111] Zehnder, B.: Collusion in online poker pays off. Bachelor's thesis, ETH Zurich, Zurich, Switzerland (2012)