Department of Computer Science
Series of Publications A
Report A-2009-5

Optimisation Problems in Wireless Sensor Networks: Local Algorithms and Local Graphs

Jukka Suomela

To be presented, with the permission of the Faculty of Science of the University of Helsinki, for public criticism in Auditorium XIV, University Main Building, on June 13th, 2009, at 10 o'clock.

University of Helsinki
Finland

Contact information

Postal address:
Department of Computer Science
P.O. Box 68 (Gustaf Hällströmin katu 2b)
FI-00014 University of Helsinki
Finland

Email address: postmaster@cs.Helsinki.FI (Internet)

URL: http://www.cs.Helsinki.FI/

Telephone: +358 9 1911

Telefax: +358 9 191 51120

ISSN 1238-8645
ISBN 978-952-10-5599-7 (paperback)
ISBN 978-952-10-5600-0 (PDF)
Computing Reviews (1998) Classification: F.2.2, C.2.4, F.1.1, G.1.6
Helsinki 2009
Helsinki University Print

Optimisation Problems in Wireless Sensor Networks: Local Algorithms and Local Graphs

Jukka Suomela

Department of Computer Science
P.O. Box 68, FI-00014 University of Helsinki, Finland
jukka.suomela@cs.helsinki.fi
http://www.cs.helsinki.fi/jukka.suomela/

PhD Thesis, Series of Publications A, Report A-2009-5
Helsinki, May 2009, xii + 106 + 96 pages
ISSN 1238-8645
ISBN 978-952-10-5599-7 (paperback)
ISBN 978-952-10-5600-0 (PDF)

Abstract

This thesis studies optimisation problems related to modern large-scale distributed systems, such as wireless sensor networks and wireless ad-hoc networks. The concrete tasks that we use as motivating examples are the following: (i) maximising the lifetime of a battery-powered wireless sensor network, (ii) maximising the capacity of a wireless communication network, and (iii) minimising the number of sensors in a surveillance application.

A sensor node consumes energy both when it is transmitting or forwarding data, and when it is performing measurements. Hence task (i), lifetime maximisation, can be approached from two different perspectives. First, we can seek for optimal data flows that make the most out of the energy resources available in the network; such optimisation problems are examples of so-called *max-min linear programs*. Second, we can conserve energy by putting redundant sensors into sleep mode; we arrive at the *sleep scheduling* problem, in which the objective is to find an optimal schedule that determines when each sensor node is asleep and when it is awake.

In a wireless network simultaneous radio transmissions may interfere with each other. Task (ii), capacity maximisation, therefore gives rise to another scheduling problem, the *activity scheduling* problem, in which the objective is to find a minimum-length conflict-free schedule that satisfies the data transmission requirements of all wireless communication links.

Task (iii), minimising the number of sensors, is related to the classical graph problem of finding a minimum dominating set. However, if we are not only interested in detecting an intruder but also locating the intruder, it is not sufficient to solve the dominating set problem; formulations such as minimum-size *identifying codes* and *locating–dominating codes* are more appropriate.

This thesis presents *approximation algorithms* for each of these optimisation problems, i.e., for max-min linear programs, sleep scheduling, activity scheduling, identifying codes, and locating–dominating codes. Two complementary approaches are taken. The main focus is on *local algorithms*, which are constant-time distributed algorithms. The contributions include local approximation algorithms for max-min linear programs, sleep scheduling, and activity scheduling. In the case of max-min linear programs, tight upper and lower bounds are proved for the best possible approximation ratio that can be achieved by any local algorithm.

The second approach is the study of centralised polynomial-time algorithms in *local graphs* – these are geometric graphs whose structure exhibits spatial locality. Among other contributions, it is shown that while identifying codes and locating–dominating codes are hard to approximate in general graphs, they admit a polynomial-time approximation scheme in local graphs.

Computing Reviews (1998) Categories and Subject Descriptors:
F.2.2 [Analysis of Algorithms and Problem Complexity]: Nonnumerical Algorithms and Problems – computations on discrete structures, sequencing and scheduling
C.2.4 [Computer-Communication Networks]: Distributed Systems
F.1.1 [Computation by Abstract Devices]: Models of Computation
G.1.6 [Numerical Analysis]: Optimization – linear programming

General Terms:
Algorithms, Theory

Additional Key Words and Phrases:
local algorithms, wireless sensor networks

Acknowledgements

I thank my supervisor, Patrik Floréen, for support and guidance. Petteri Kaski and Valentin Polishchuk deserve special thanks for collaboration and numerous discussions.

I thank my coauthors, including Michael A. Bender, Alon Efrat, Sándor P. Fekete, Poornananda R. Gaddehosur, Marja Hassinen, Joel Kaasinen, Jukka Kohonen, Evangelos Kranakis, Alexander Kröller, Joonas Kukkonen, Eemil Lagerspetz, Sian Lun Lau, Vincenzo Liberatore, Miquel Martin, Jean Millerat, Joseph S. B. Mitchell, Topi Musto, Petteri Nurmi, Aleksi Penttinen, Remco Poortinga, Alfons Salden, Michael Sutterer, and Andreas Wiese, for fruitful collaboration.

I am grateful to my pre-examiners Keijo Heljanko and Christian Scheideler for their comments and suggestions. I also thank everyone else who has given me advice and feedback on my work, including Jyrki Kivinen, Mikko Koivisto, Marina Kurtén, Lauri Malmi, Pekka Orponen, Satu Elisa Schaeffer, Roger Wattenhofer, and numerous anonymous referees. Finally, many thanks to Tanja Säily for her encouragement.

This work was supported in part by the Academy of Finland, Grants 116547 and 202203, by the IST Programme of the European Community, under the PASCAL Network of Excellence, IST-2002-506778, by Helsinki Graduate School in Computer Science and Engineering (Hecse), and by the Foundation of Nokia Corporation.

Contents

Original publications

This thesis is based on the following original publications, which are referred to in the text as Papers I–VI. The papers are reprinted at the end of this thesis.

I. Patrik Floréen, Petteri Kaski, Topi Musto, and Jukka Suomela. Approximating max-min linear programs with local algorithms. In *Proc. 22nd IEEE International Parallel and Distributed Processing Symposium (IPDPS, Miami, FL, USA, April 2008)*. IEEE, Piscataway, NJ, USA, 2008.

II. Patrik Floréen, Marja Hassinen, Petteri Kaski, and Jukka Suomela. Tight local approximation results for max-min linear programs. In *Proc. 4th International Workshop on Algorithmic Aspects of Wireless Sensor Networks (Algosensors, Reykjavík, Iceland, July 2008)*, volume 5389 of *Lecture Notes in Computer Science*, pages 2–17. Springer, Berlin, Germany, 2008.

III. Patrik Floréen, Petteri Kaski, and Jukka Suomela. A distributed approximation scheme for sleep scheduling in sensor networks. In *Proc. 4th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON, San Diego, CA, USA, June 2007)*, pages 152–161. IEEE, Piscataway, NJ, USA, 2007.

IV. Patrik Floréen, Petteri Kaski, Topi Musto, and Jukka Suomela. Local approximation algorithms for scheduling problems in sensor networks. In *Proc. 3rd International Workshop on Algorithmic Aspects of Wireless Sensor Networks (Algosensors, Wrocław, Poland, July 2007)*, volume 4837 of *Lecture Notes in Computer Science*, pages 99–113. Springer, Berlin, Germany, 2008.

V. Petteri Kaski, Aleksi Penttinen, and Jukka Suomela. Coordinating concurrent transmissions: A constant-factor approximation of maximum-weight independent set in local conflict graphs. *Ad Hoc & Sensor Wireless Networks: An International Journal*, 6(3–4):239–263, 2008.

VI. Jukka Suomela. Approximability of identifying codes and locating–dominating codes. *Information Processing Letters*, 103(1):28–33, 2007.

Chapter 1

Introduction

This thesis studies computational problems motivated by distributed systems, in particular by *wireless sensor networks* [3, 38, 101, 177].

A wireless sensor network consists of a number of *sensor nodes*. As the name suggests, each sensor node is equipped with one or more sensors that produce measurements of the environment. Other components of a sensor node include a small computer, an energy source such as a battery, and a radio transmitter–receiver (transceiver).

In typical applications, the measurements need to be gathered in a central location – a *sink node* – for further analysis and processing. To help with data gathering, we may install separate *relay nodes* which are responsible for relaying data between the sensor nodes and the base station.

Figure 1.1 shows a schematic example of a simple sensor network, with a sink node, three relay nodes, and five sensor nodes. This is a two-tier network: the first tier of the network consists of low-power short-range radio transmissions between sensor nodes and nearby relay nodes, and the second tier consists of high-power long-distance radio transmissions between the relay nodes and the sink node.

Uses of sensor networks range from scientific applications such as volcano monitoring [167] to industrial applications such as monitoring vibrations in an oil tanker [2]. But regardless of the application, it is desirable to minimise the cost of the system, as well as to make the most out of the system that we have installed. These requirements give rise to various optimisation problems. In this thesis, we focus on three examples of concrete tasks: (i) maximising the lifetime of a battery-powered wireless sensor network, (ii) maximising the capacity of a wireless communication network, and (iii) minimising the number of sensors in a surveillance application.

To give an example of task (i), lifetime maximisation, consider the network in Figure 1.1. Several different choices of data flows are possible; for

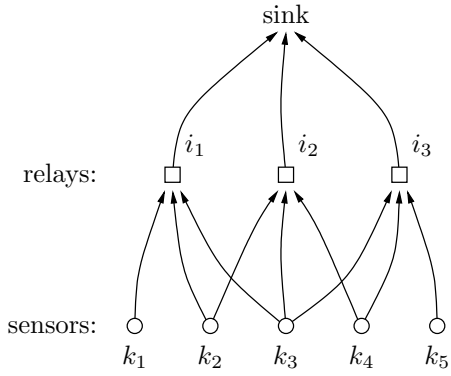

Figure 1.1: A sensor network. The arrows illustrate possible paths for transmitting data from the sensor nodes to the sink node

example, the sensors k_1, k_2, and k_3 could forward all data via the relay i_1, the sensor k_4 could forward all data via i_2, and the sensor k_5 could forward all data via i_3. However, in that case the relay i_1 would need to forward three times as much data as the relay i_2 or i_3, which also means that the battery of the relay i_1 would drain faster than the batteries of other relays. In this simple network, it is easy to find data flows that use the relays more evenly and therefore provide a longer lifetime for the network, but in a larger network the problem becomes challenging. As we shall see, the problem of finding *optimal* data flows that maximise the lifetime of the network can be formulated as a linear program; in particular, such linear programs are examples of *max-min linear programs*. We study algorithms for solving max-min linear programs in Chapter 3.

Another example of a lifetime maximisation problem can be found in Chapter 4, in which we study the *sleep scheduling* problem. For example, if the sensors k_1 and k_2 in Figure 1.1 are physically very close to each other, then the measurements from k_1 and k_2 are likely to be highly correlated, and we only need to have measurements from one of them. Therefore we can conserve energy by putting the sensor k_1 into sleep mode whenever the sensor k_2 is awake and vice versa. In the sleep scheduling problem, the objective is to find an optimal schedule that determines when each sensor node is asleep and when it is awake.

An example of task (ii), capacity maximisation, is given in Chapters 4 and 5, in which we study the *activity scheduling* problem. In a wireless network, simultaneous data transmissions may interfere with each other. In the activity scheduling problem, the objective is to find a minimum-length

interference-free schedule that satisfies the data transmission requirements of all wireless communication links. Finally, we will see an example of task (iii), minimising the number of sensors, in Chapter 5 when we study so-called *identifying codes* and *locating–dominating codes*.

While task (iii) is primarily related to the *planning* of a network deployment, tasks (i) and (ii) are related to the *operation* of a network that has already been installed. To control the operation of a network, we need a *distributed algorithm* that we can run in the network: the nodes of the network are computational entities, the network itself is the input for the algorithm, and the nodes use the output of the algorithm to control their own operation – for example, to decide which relay to use for forwarding data towards the sink, or to decide when to enter sleep mode and when to wake up.

In a small network, such as the one illustrated in Figure 1.1, a central entity such as the sink node can gather full information on the network, solve the optimisation problem by using a centralised algorithm, and send the relevant part of the solution to each node. As the size of the network increases, such an approach becomes infeasible. Therefore the main focus of this thesis is on highly scalable distributed algorithms. In particular, we study *local algorithms*, i.e., constant-time distributed algorithms.

We begin this thesis in Chapter 2 by giving an introduction to local algorithms and their advantages and applications; we also survey the known positive and negative results on local algorithms. Chapters 3 and 4 provide new examples of local algorithms.

The algorithms that we study in Chapter 5 take a complementary perspective. These are traditional centralised polynomial-time algorithms (as opposed to distributed constant-time algorithms) that we can use, for example, when we are planning a network deployment. The problems that we study are hard to solve exactly or approximately in general graphs; however, typical sensor networks are hardly similar to the pathological constructions familiar from inapproximability proofs. We focus on more restrictive families of graphs that can be used to model realistic wireless sensor networks. In particular, we study so-called *local graphs* and *local conflict graphs*, which are geometric graphs whose structure exhibits spatial locality. We present a polynomial-time constant-factor approximation algorithm for activity scheduling in local conflict graphs, and a polynomial-time approximation scheme for identifying codes and locating–dominating codes in local graphs.

Chapter 6 concludes this thesis with possible directions for future research.

1.1 Main results of the original publications

Papers I–IV focus on local algorithms, i.e., constant-time distributed algorithms (see Chapter 2 for background). In particular, Papers I and II study local algorithms for *max-min linear programs* (see Chapter 3). In a max-min linear program, the goal is to maximise the minimum over several non-negative linear objective functions, subject to non-negative linear constraints. Paper I presents an approximation algorithm for a family of bounded-growth graphs. Paper II provides a tight characterisation of the local approximability of so-called bipartite max-min linear programs.

Papers III and IV study *sleep scheduling* and *activity scheduling* with local algorithms (see Chapter 4). The sleep scheduling problem is a generalisation of the *fractional domatic partition* problem, and the activity scheduling problem is a generalisation of the *fractional graph colouring* problem. The main contribution is the introduction of graphs with *markers*; a small amount of symmetry-breaking information turns out to be sufficient to design local approximation algorithms for these scheduling problems for a broad family of graphs.

Papers V and VI analyse *local conflict graphs* and *local graphs* (see Section 5.1). Paper V studies activity scheduling in a centralised setting (see Section 5.2). We show that there are polynomial-time constant-factor approximation algorithms for the maximum-weight independent set problem and the activity scheduling problem in local conflict graphs. However, there is no polynomial-time approximation scheme unless P = NP. Paper VI studies the problems of finding minimum-size *identifying codes* and *locating–dominating codes* (see Section 5.3). This work shows that it is possible to approximate both problems within a logarithmic factor, but sublogarithmic approximation ratios are intractable in general graphs. However, the problem is considerably easier to approximate in local graphs: there is a polynomial-time approximation scheme.

1.2 Contributions of the author

In Papers I–IV, the main results are due to the present author. Write-up, illustrations, technical details, and motivating examples are joint work.

In Paper V, the concept of local conflict graphs and most results are due to the present author. The results in Section 7 of Paper V are due to P. Kaski. A. Penttinen was the domain expert, and the original idea of studying these scheduling problems was due to him and his colleagues.

The writer of this thesis is the sole author of Paper VI.

Chapter 2

Local algorithms

This chapter is an introduction to local algorithms and a survey of prior work in the field. A manuscript based on this chapter has been submitted for publication [159].

This chapter also defines the terminology and notation related to graphs and distributed algorithms that we use throughout this thesis. For easier reference, commonly used symbols are also summarised in the index.

2.1 Introduction

A local algorithm is a distributed algorithm that runs in a constant number of synchronous communication rounds. Put otherwise, the output of a node in a local algorithm is a function of the input available within a constant-radius neighbourhood of the node.

Research on local algorithms was pioneered by Angluin [6], Linial [121], and Naor and Stockmeyer [138]. Angluin [6] studied the limitations of anonymous networks without any unique identifiers. Linial [121] proved seminal negative results for the case where each node has a unique identifier. Naor and Stockmeyer [138] presented the first nontrivial positive results.

We focus on algorithms whose running time and performance guarantees are independent of the number of nodes in the network – put simply, these are algorithms that could be used to control infinitely large networks in finite time. For a more general discussion on distributed algorithms, see, for example, Peleg [145] and Elkin [48].

Many of the negative results cited in this survey were not originally stated as negative results for local algorithms; they are more general results which, as a corollary, imply that a particular problem cannot be solved by any local algorithm. The emphasis is on the results that have nontrivial im-

plications on local algorithms. For more impossibility results for distributed algorithms, see the surveys by Lynch [129] and Fich and Ruppert [52].

We begin with some essential definitions in Section 2.2. We review the advantages and applications of local algorithms in Section 2.3. Section 2.4 summarises the computational problems that we study. Section 2.5 discusses what information each node has available in a local algorithm. Section 2.6 reviews negative results: what cannot be computed with a local algorithm. Section 2.7 reviews positive results: what deterministic local algorithms are known. In Section 2.8 we study the power of randomness, in comparison with deterministic local algorithms. In Section 2.9 we study local algorithms in a geometric setting, in which each node knows its coordinates. Section 2.10 concludes this survey with some open problems.

For a quick summary of the negative results for deterministic local algorithms, see Tables 2.1 and 2.2 on pages 25–26. The positive results for deterministic local algorithms are summarised in Tables 2.3 and 2.4 on pages 30–31. Many of the results summarised in the tables are corollaries that have not been stated explicitly in the literature.

2.2 Definitions

In this work, all graphs are simple and undirected unless otherwise mentioned. For a graph $\mathcal{G} = (V, E)$, we use the following notation and terminology. An undirected edge between the nodes $u \in V$ and $v \in V$ is represented by an unordered pair $\{u, v\} \in E$. We write $\deg(v)$ for the degree (number of neighbours) of the node $v \in V$. A node $v \in V$ is isolated if $\deg(v) = 0$. The graph $\mathcal{G}$ is k-regular if $\deg(v) = k$ for each $v \in V$.

We use $d_{\mathcal{G}}(u, v)$ to denote the shortest-path distance (number of edges, hop count) between the nodes u and v in the graph $\mathcal{G}$, and $B_{\mathcal{G}}(v, r) = \{u \in V : d_{\mathcal{G}}(u, v) \leq r\}$ to denote the radius-r neighbourhood of the node v in $\mathcal{G}$. We write $\mathcal{G}(v, r)$ for the subgraph of $\mathcal{G}$ induced by $B_{\mathcal{G}}(v, r)$. We occasionally refer to the subgraph $\mathcal{G}[v, r]$ of $\mathcal{G}(v, r)$; the graph $\mathcal{G}[v, r]$ is constructed from $\mathcal{G}(v, r)$ by removing the edges $\{s, t\}$ with $d_{\mathcal{G}}(v, s) = d_{\mathcal{G}}(v, t) = r$.

The graph $\mathcal{G} = (V, E)$ is bipartite if $V = V_1 \cup V_2$ for disjoint sets V_1 and V_2 such that each edge $e \in E$ is of the form $e = \{u, v\}$ for $u \in V_1$ and $v \in V_2$. The complete graph on n nodes is denoted by K_n. Terminology related to directed graphs is introduced in Section 2.5.3. Geometric graphs such as unit-disk graphs are defined in Section 2.9.1.

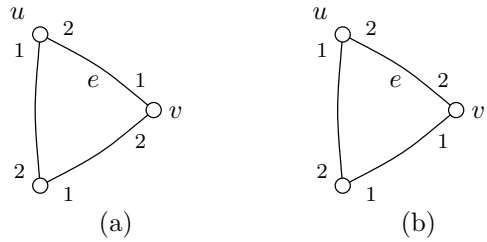

Figure 2.1: A communication graph with a port numbering.

2.2.1 Communication graph

Throughout this work, the graph $\mathcal{G} = (V, E)$ is the communication graph of a distributed system: each node $v \in V$ is a computational entity and an edge $\{u, v\} \in E$ denotes that the nodes u and v can communicate with each other.

Often we have to make assumptions on the structure of the communication graph. Among others, we study the family of *bounded-degree graphs*. In this case we assume that there is a known constant Δ, and any node in any communication graph $\mathcal{G}$ that we may encounter is guaranteed to have at most Δ neighbours.

2.2.2 Port numbering

We assume that there is a port numbering (local edge labelling) [6, 12, 174] available for the communication graph $\mathcal{G}$. This means that each node of $\mathcal{G}$ imposes an ordering on its incident edges. Thus each edge $\{u, v\} \in E$ has two natural numbers associated with it: the port number in the node u, denoted by $p(u, v)$, and the port number in the node v, denoted by $p(v, u)$. If $p(u, v) = i$, we also say that the neighbour i of u is v.

See Figure 2.1 for an illustration. Figure 2.1a shows one possible way to assign the port numbers in a 3-cycle. The port number in the node u for the edge $e = \{u, v\} \in E$ is $p(u, v) = 2$, and the port number in v for e is $p(v, u) = 1$. The neighbour 2 of u is v. Figure 2.1b shows another way to assign the port numbers in the same graph.

2.2.3 Model of distributed computing

We use Linial's [121] model of computation; Peleg [145] calls it the *local* model. Each node in the system executes the same algorithm $\mathcal{A}$. Initially, each node $v \in V$ knows a task-specific *local input* i_v. Each node $v \in V$ has

to produce a *local output* o_v. We always assume that $\deg(v)$ is part of the local input i_v. The local input i_v may also contain auxiliary information such as unique node identifiers; this is discussed in more detail in Section 2.5.

The distributed system operates in a synchronous manner. Let r be the number of synchronous communication rounds. In each round $i = 1, 2, \ldots, r$, the following operations are performed, in this order:

1. Each node performs local computation.
2. Each node v sends one message to each port $1, 2, \ldots, \deg(v)$.
3. Each node v receives one message from each port $1, 2, \ldots, \deg(v)$.

Finally, after the round r, each node $v \in V$ performs local computation and announces its local output o_v. The size of a message is unbounded and local computation is free.

Example 2.1. Consider the graph in Figure 2.1a. If the node u sends a message m to the port 2 in the round i, the same message m is received by the node v from the port 1 in the same round i. Note that the node u can include the outgoing port number 2 in the message m; then the receiver v learns that the edge number 1 in v equals the edge number 2 in its neighbour u.

2.2.4 Local algorithm and local horizon

We say that $\mathcal{A}$ is a *local algorithm* if the number of communication rounds r is a constant. The constant r may depend on the parameters of the problem family; for example, if we study bounded-degree graphs, the value of r may depend on the parameter Δ. However, the value of r cannot depend on the problem instance; in particular, it does not depend on the number of nodes in the graph $\mathcal{G}$.

The constant r is called the *local horizon* of the local algorithm. In r synchronous communication rounds, information can be propagated for exactly r hops in the network. The output o_v of the node $v \in V$ may depend on the local inputs i_u for all $u \in B_{\mathcal{G}}(v, r)$; however, it cannot depend on the local inputs i_u for any $u \notin B_{\mathcal{G}}(v, r)$. A decision must be made based on information available within the local horizon.

Throughout this work, the original definition of a local algorithm by Naor and Stockmeyer [138] is used: r must be a constant. In many papers, the term "local algorithm" is used in a less strict manner, and terminology such as "strictly local algorithm" or "$O(1)$-local algorithm" is used to refer to the case of a constant r.

2.2.5 Local approximation

An *α-approximation algorithm* is an algorithm that produces a feasible output, and the utility of the output is guaranteed to be within factor α of the utility of an optimal solution. We use the convention that $\alpha \geq 1$ for both minimisation and maximisation problems [13]. Hence, for a minimisation problem, an α-approximation algorithm produces a feasible solution with a cost at most $\alpha \cdot \mathrm{OPT}$ where OPT is the cost of an optimal solution, and for a maximisation problem, an α-approximation algorithm produces a feasible solution with the utility at least OPT/α where OPT is the utility of an optimal solution.

A *local α-approximation algorithm* is an α-approximation algorithm and a local algorithm. A *local approximation scheme* is a family of local algorithms such that for each $\varepsilon > 0$ there is a local $(1 + \varepsilon)$-approximation algorithm.

2.2.6 Distributed constant-size problem

We say that a problem has *distributed constant size* if $\mathcal{G}$ is a bounded-degree graph and the size of the local input i_v is bounded by a constant. If we have a local algorithm for a distributed constant-size problem, then each node needs to transmit and process only a constant number of bits; therefore also local computations can be done in constant time, and the size of the local output o_v is bounded by a constant as well. Informally, a local algorithm for a distributed constant-size problem runs in constant time, regardless of the details of the model of distributed computing; we do not need to exploit unbounded size of messages and unlimited local computation.

2.3 Advantages and applications

2.3.1 Fault tolerance and robustness

A local algorithm is not only highly scalable but also fault-tolerant. A local algorithm recovers efficiently from failures, changes in the network topology, and changes in the input [138]. If the input of a node $v \in V$ changes, this only affects the output within $B_{\mathcal{G}}(v, r)$; Mayer et al. [130] formalise this notion of robustness.

We can say that a local algorithm is a *dynamic graph algorithm* [49], that is, it can be used to maintain a feasible solution in a dynamic graph

in which edges and nodes are added and deleted. In the case of distributed constant-size problems, a local algorithm supports arbitrary updates in the graph in constant time per operation.

Naor and Stockmeyer [138] point out the connection between local algorithms and *self-stabilising algorithms* [44, 45, 155]. A self-stabilising algorithm arrives at a legitimate state – "stabilises" – in finite time regardless of the initial states of the nodes. Work on self-stabilising algorithms [15, 16] provides, as a simple special case, a mechanical way to transform a constant-time deterministic distributed algorithm into a self-stabilising algorithm that stabilises in constant time. Therefore local algorithms for distributed constant-size problems are not only self-stabilising but also self-organising [46].

2.3.2 Value of information

In the model of local algorithms, we assume that local computation is free. Hence our focus is primarily on the amount of information needed in distributed decision making: what can we do with the information that is available in the constant-radius neighbourhood of a node. Positive and negative results for local algorithms can be interpreted as information-theoretic upper and lower bounds; they give insight into the value of information [141, 143].

2.3.3 Other models of computing

Local algorithms are closely connected to circuit complexity and the complexity class NC^0 [1]: if a distributed constant-size problem can be solved with a local algorithm, then for any bounded-degree graph $\mathcal{G}$ there is a bounded-fan-in Boolean circuit that maps the local inputs to the local outputs, and the depth of the circuit is independent of the size of $\mathcal{G}$. As pointed out by Wattenhofer and Wattenhofer [164], a local algorithm provides an efficient algorithm in the PRAM model, but a PRAM algorithm is not necessarily local.

Sterling [158] shows that lower bounds for local algorithms can be applied to derive lower bounds in the tile-assembly model. Gibbons [62] points out that the envisioned shape-shifting networks will require not only advances in hardware, but also novel local algorithms.

2.3.4 Sublinear-time centralised algorithms

A local algorithm for a distributed constant-size problem provides a linear-time centralised algorithm: simply simulate the local algorithm for each node. Parnas and Ron [144] show that in some cases it is possible to

use a local algorithm to design a sublinear-time (or even constant-time) centralised approximation algorithm; see also Nguyen and Onak [139] and Floréen et al. [55].

For example, consider a local approximation algorithm $\mathcal{A}$ for the vertex cover problem (see Section 2.4.1 below for the definition). For a given input graph $\mathcal{G}$, the algorithm $\mathcal{A}$ produces a feasible and approximately optimal vertex cover C. From the point of view of a centralised algorithm, the local algorithm $\mathcal{A}$ can be interpreted as an oracle with which we can access the cover C: for any given node v, we can efficiently determine whether $v \in C$ or not by simulating the local algorithm $\mathcal{A}$ at the node v. Therefore we can estimate the size of C by sampling nodes uniformly at random; for each node we determine whether it is in C or not. Furthermore, as we know that C is approximately optimal, estimating the size of C allows us to estimate the size of the minimum vertex cover of the graph $\mathcal{G}$ as well.

These kinds of algorithms can be used to obtain information about the global properties of very large graphs. Lovász [124] gives examples of such graphs: the Internet, the social network of all living people, the human brain, and crystal structures. Many of these are not explicitly given and not completely known; however, it may be possible to obtain information about these graphs by sampling nodes and their local neighbourhoods. From this perspective, the sublinear-time algorithm by Parnas and Ron [144] puts together neighbourhood sampling and a local approximation algorithm to estimate the global properties of huge graphs.

2.4 Problems

Now we proceed to give the definitions of the computational problems that we discuss in this survey. Most of these problems are classical combinatorial problems; for details and background, see textbooks on graph theory [43], combinatorial optimisation [98, 140], NP-completeness [60], and approximation algorithms [13, 162].

When we study local algorithms, we assume that the problem instance is given in a distributed manner: each node in the communication graph $\mathcal{G}$ knows part of the input. For graph problems, the connection between the communication graph $\mathcal{G}$ and the structure of the problem instance is usually straightforward: we simply assume that the communication graph $\mathcal{G}$ is our input graph. For more general packing and covering problems (such as the set cover problem or packing LPs) there is more freedom. However, it is fairly natural to represent such problems in terms of bipartite graphs, and

this has been commonly used in literature [19, 108, 143]. We follow this convention.

The exact definitions of the local input i_v and output o_v are usually fairly straightforward. For unweighted graph problems, we do not need any task-specific information in the local input i_v; the structure of the communication graph $\mathcal{G}$ is enough. For weighted graph problems, the local input i_v contains the weight of the node v and the weights of the incident edges. Hence all unweighted graph problems in bounded-degree graphs are distributed constant-size problems; weighted problems are distributed constant-size problems if the weights are represented with a bounded number of bits.

If the output is a subset $X \subseteq V$ of nodes, then the local output o_v is simply one bit of information: whether $v \in X$ or not. If the output is a subset $X \subseteq E$ of edges, then the local output o_v contains one bit for each incident edge e. The algorithm must produce a correct output no matter how we choose the port numbers in the communication graph $\mathcal{G}$.

2.4.1 Graph problems

A set of nodes $I \subseteq V$ is an *independent set* if no two nodes in I are adjacent, that is, there is no edge $\{u, v\} \in E$ with $u \in I$ and $v \in I$. An independent set I is *maximal* if it cannot be extended, that is, $I \cup \{v\}$ is not an independent set for any $v \in V \setminus I$.

A set of edges $M \subseteq E$ is a *matching* if the edges in M do not share a node, that is, if $\{t, u\} \in M$ and $\{t, v\} \in M$ then $u = v$. Again, a matching is *maximal* if it cannot be extended. A set of edges $M \subseteq E$ is a *simple 2-matching* if for each node u, the number of edges $e \in M$ with $u \in e$ is at most 2.

Let $M \subseteq E$ be a matching in a bipartite graph. An edge $\{u, v\} \in E \setminus M$ is *unstable* if $\{u, s\} \in M$ implies $p(u, s) > p(u, v)$ and $\{v, t\} \in M$ implies $p(v, t) > p(v, u)$. That is, if we interpret port numbers as a ranking of possible partners, both u and v would prefer each other to their current partners (if any). The matching M is *ε-stable* [55] if the number of unstable edges is at most $\varepsilon|M|$, and the matching is *stable* [59, 68] if there is no unstable edge.

A set of edges $C \subseteq E$ is an *edge cover* if for each node $v \in V$ there is an edge $e \in C$ with $v \in e$. An edge cover exists if and only if there is no isolated node. A set of nodes $C \subseteq V$ is a *vertex cover* if $V \setminus C$ is an independent set. In other words, C is a vertex cover if for each edge $\{u, v\} \in E$ either $u \in C$ or $v \in C$ or both.

A set of nodes $D \subseteq V$ is a *dominating set* if every node $v \in V$ has $v \in D$ or there is a neighbour u of v with $u \in D$. A maximal independent set is a dominating set. A dominating set D is *connected* if the subgraph of $\mathcal{G}$ induced by D is connected. A *domatic partition* [32, 51] is a partition of V into disjoint dominating sets.

A set of edges $D \subseteq E$ is an *edge dominating set* [29, 57, 175] if for each edge $\{u, v\} \in E$ there is an edge $e \in D$ incident to u or v or both. A maximal matching is an edge dominating set.

A *cut* is a partition of V into two sets, $V = X \cup Y$. The size of the cut is the number of edges $\{u, v\} \in E$ with $u \in X$ and $v \in Y$.

A *vertex k-colouring* of $\mathcal{G}$ assigns a colour from the set $\{1, 2, \ldots, k\}$ to each node of $\mathcal{G}$ such that adjacent nodes have different colours. An *edge colouring* is analogous: one assigns a colour to each edge, such that adjacent edges have different colours.

Naor and Stockmeyer [138] define the problem of *weak colouring*. A weak k-colouring of $\mathcal{G}$ assigns labels $\{1, 2, \ldots, k\}$ to the nodes of $\mathcal{G}$ such that each non-isolated node must have at least one neighbour with a different label. A graph admits a vertex 2-colouring if and only if it is bipartite; however, any graph admits a weak 2-colouring.

The *maximum matching problem* refers to the optimisation problem of finding a matching M that maximises $|M|$. The maximum independent set problem, the (minimum) vertex cover problem, the (minimum) dominating set problem, the maximum cut problem, the minimum cut problem, etc., are analogous.

2.4.2 Covering problems

Let $\mathcal{G} = (V \cup K, E)$ be a bipartite graph. Each edge $\{v, k\} \in E$ joins an *agent* $v \in V$ and a *customer* $k \in K$; each node knows its role. The maximum degree of an agent $v \in V$ is Δ_V, and the maximum degree of a customer $k \in K$ is Δ_K. A subset $X \subseteq V$ is a *set cover* if each customer is covered by at least one agent in X, that is, for each customer $k \in K$ there is an adjacent agent $v \in X$ with $\{k, v\} \in E$. See Figure 2.2a.

The vertex cover problem is a special case of the set cover problem with $\Delta_K = 2$: each customer (edge) can be covered by 2 agents (nodes). The edge cover problem is a special case of the set cover problem with $\Delta_V = 2$: each agent (edge) covers 2 customers (nodes). The dominating set problem in a graph with maximum degree Δ is a special case of the set cover problem with $\Delta_V = \Delta_K = \Delta + 1$.

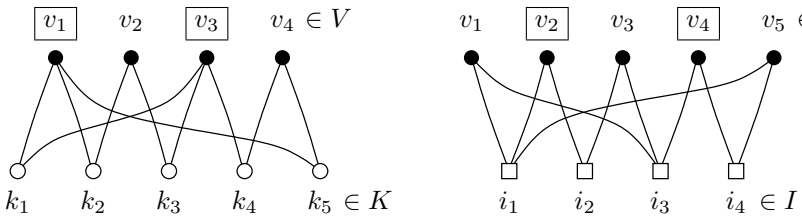

Figure 2.2: (a) A set cover instance with $\Delta_V = 3$ and $\Delta_K = 2$; a minimum-size solution is $X = \{v_1, v_3\}$. (b) A set packing instance with $\Delta_V = 2$ and $\Delta_I = 3$; a maximum-size solution is $X = \{v_2, v_4\}$.

The problem of finding a minimum-size set cover can be written as an integer program

$$\begin{aligned} \text{minimise} \quad & \sum_{v \in V} x_v \\ \text{subject to} \quad & \sum_{v \in V} c_{kv} x_v \geq 1 \qquad \forall\, k \in K, \\ & x_v \in \{0,1\} \quad \forall\, v \in V, \end{aligned} \tag{2.1}$$

where $c_{kv} = 0$ if $\{k, v\} \notin E$ and $c_{kv} = 1$ if $\{k, v\} \in E$. The LP relaxation of (2.1) is a 0/1 *covering LP*

$$\begin{aligned} \text{minimise} \quad & \sum_{v \in V} x_v \\ \text{subject to} \quad & \sum_{v \in V} c_{kv} x_v \geq 1 \quad \forall\, k \in K, \\ & x_v \geq 0 \quad \forall\, v \in V. \end{aligned} \tag{2.2}$$

In a general *covering LP* we can have an arbitrary $c_{kv} \geq 0$ for each edge $\{k, v\} \in E$.

2.4.3 Packing problems

Let $\mathcal{G} = (V \cup I, E)$ be a bipartite graph. Each edge $\{v, i\} \in E$ joins an *agent* $v \in V$ and a *constraint* $i \in I$; each node knows its role. The maximum degree of an agent $v \in V$ is Δ_V, and the maximum degree of a constraint $i \in I$ is Δ_I. A subset $X \subseteq V$ is a *set packing* if each constraint is covered by at most one agent in X, that is, for each constraint $i \in I$ there is at most one adjacent agent $v \in X$ with $\{v, i\} \in E$. See Figure 2.2b.

The independent set problem is a special case of the set packing problem with $\Delta_I = 2$. The maximum matching problem is a special case of the set packing problem with $\Delta_V = 2$.

The problem of finding a maximum-size set packing can be written as an integer program

$$\begin{aligned} \text{maximise} \quad & \sum_{v \in V} x_v \\ \text{subject to} \quad & \sum_{v \in V} a_{iv} x_v \leq 1 \qquad \forall\, i \in I, \\ & x_v \in \{0,1\} \quad \forall\, v \in V, \end{aligned} \tag{2.3}$$

where $a_{iv} = 0$ if $\{i, v\} \notin E$ and $a_{iv} = 1$ if $\{i, v\} \in E$. The LP relaxation of (2.3) is a 0/1 *packing LP*

$$\begin{aligned} \text{maximise} \quad & \sum_{v \in V} x_v \\ \text{subject to} \quad & \sum_{v \in V} a_{iv} x_v \leq 1 \quad \forall\, i \in I, \\ & x_v \geq 0 \quad \forall\, v \in V. \end{aligned} \tag{2.4}$$

In a general *packing LP* we can have an arbitrary $a_{iv} \geq 0$ for each $\{i, v\} \in E$. A packing LP is a dual of a covering LP and vice versa.

2.4.4 Mixed packing and covering

Finally, we can study linear programs with both packing constraints (constraints of the form $A\mathbf{x} \leq \mathbf{1}$ for a non-negative matrix A) and covering constraints (constraints of the form $C\mathbf{x} \geq \mathbf{1}$ for a non-negative matrix C). In general, it may be that there is no feasible solution that satisfies both packing and covering constraints; however, we can formulate a linear program where the objective is to violate the covering constraints as little as possible (the case of violating the packing constraints as little as possible is analogous). We arrive at a *max-min LP* where the objective is to maximise ω subject to $A\mathbf{x} \leq \mathbf{1}$, $C\mathbf{x} \geq \omega\mathbf{1}$, and $\mathbf{x} \geq \mathbf{0}$.

In a distributed setting, we have a bipartite graph $\mathcal{G} = (V \cup I \cup K, E)$. Each edge $e \in E$ is of the form $e = \{v, i\}$ or $e = \{v, k\}$ where $v \in V$ is an *agent*, $i \in I$ is a *constraint*, and $k \in K$ is a *customer* (or an *objective*); each node knows its role. The maximum degree of an agent $v \in V$ is Δ_V,

the maximum degree of a constraint $i \in I$ is Δ_I, and the maximum degree of a customer $k \in K$ is Δ_K. The objective is to

$$\begin{aligned} &\text{maximise} && \omega \\ &\text{subject to} && \sum_{v \in V} a_{iv} x_v \leq 1 && \forall\, i \in I, \\ &&& \sum_{v \in V} c_{kv} x_v \geq \omega && \forall\, k \in K, \\ &&& x_v \geq 0 && \forall\, v \in V. \end{aligned} \tag{2.5}$$

Again, $a_{iv} = 0$ if $\{i, v\} \notin E$, $a_{iv} \geq 0$ if $\{i, v\} \in E$, $c_{kv} = 0$ if $\{k, v\} \notin E$, and $c_{kv} \geq 0$ if $\{k, v\} \in E$. In a 0/1 *max-min LP* we have $a_{iv}, c_{kv} \in \{0, 1\}$ for all $i \in I$, $k \in K$, and $v \in V$.

2.5 Auxiliary information and local views

So far we have not assumed that the local algorithm has access to any information beyond the port numbering and the task-specific local input i_v. If we do not have any auxiliary information such as unique node identifiers in i_v, we call the network *anonymous*.

In an anonymous network, a port numbering does not provide enough information to break the symmetry [6, 88, 174]. Consider a deterministic local algorithm and the network in Figure 2.1a on page 7. The port-numbered graph is symmetric. It is easy to see that we cannot break the symmetry with the local algorithm if the local inputs are identical. Whatever message the node u sends to its port $x \in \{1, 2\}$ on the first communication round, the node v sends the same message to its port x if both run the same deterministic algorithm. Whatever message the node u receives from its port x on the first communication round, the node v receives the same message from its port x. The local state of the node u after r communication rounds is equal to the local state of the node v after r communication rounds. Eventually, the local output o_u is identical to the local output o_v.

More generally, we can choose the port numbers in an n-cycle so that for each node the port number 1 leads in a counterclockwise direction and the port number 2 leads in a clockwise direction. If the local inputs are identical, the local outputs are identical as well, regardless of the local horizon r. From the point of view of most combinatorial problems, this is discouraging: an empty set is the only matching or independent set that can be constructed by any local algorithm in this case; the set of all nodes is the only dominating set or vertex cover that can be constructed; and vertex colouring or edge colouring is not possible.

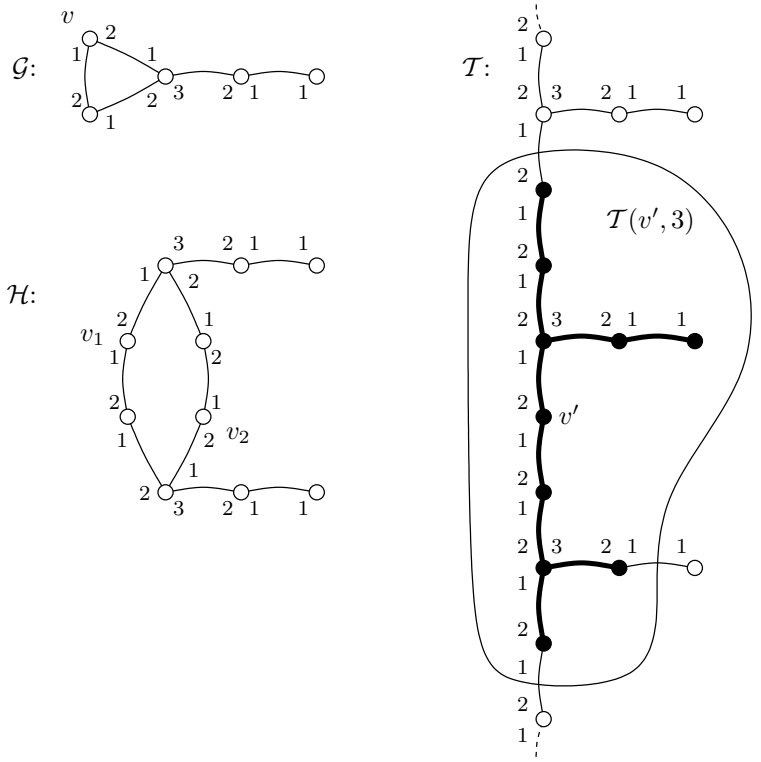

Figure 2.3: Covering graphs.

However, there are some positive examples of local algorithms that do not require any auxiliary information besides a port numbering. To better understand the possibilities and limitations of this model, we first introduce the concepts of covering graphs and unfoldings.

2.5.1 Covering graphs and unfoldings

We say that a port-numbered graph $\mathcal{H} = (V_\mathcal{H}, E_\mathcal{H})$ is a *covering graph* of $\mathcal{G} = (V, E)$ if there is a surjective mapping $f\colon V_\mathcal{H} \to V$ with the following property: for each $v \in V_\mathcal{H}$ and for each integer x, the neighbour x of v in $\mathcal{H}$ is u if and only if the neighbour x of $f(v)$ in $\mathcal{G}$ is $f(u)$. The surjection f is a *covering map*. See Figure 2.3 for an illustration: the graph $\mathcal{H}$ is a covering graph of $\mathcal{G}$; we can choose a covering map f with $f(v_1) = f(v_2) = v$.

The *unfolding* or the universal covering graph [6] of a connected graph $\mathcal{G}$ is an acyclic, connected covering graph $\mathcal{T}$. The unfolding always exists,

it is unique (up to isomorphism), and it is finite if and only if $\mathcal{G}$ is a tree. See Figure 2.3 for an illustration: the infinite tree $\mathcal{T}$ is the unfolding of $\mathcal{G}$; we can choose a covering map f with $f(v') = v$. The tree $\mathcal{T}$ is also the unfolding of $\mathcal{H}$; we can choose, for example, a covering map f with $f(v') = v_1$. This is no coincidence; because $\mathcal{H}$ and $\mathcal{G}$ have a common covering graph – in this case $\mathcal{H}$ – they also have the same unfolding.

Informally, we can construct the unfolding $\mathcal{T}$ of a graph $\mathcal{G}$ as follows. Choose an arbitrary node of $\mathcal{G}$ as a starting point. Traverse the graph $\mathcal{G}$ in a breadth-first manner; if we revisit a node because of a cycle, treat it as a new node.

This simple intuitive explanation of the unfolding is sufficient for our purposes. See, e.g., Godsil and Royle [63, §6.8] for more information on covering graphs in a pure graph-theoretic setting; note that the term "lift" has also been used to refer to a covering graph [4, 79]. For more information on universal covering graphs, see, e.g., Angluin [6]. An analogous concept in topology is a universal covering space, see, e.g., Hocking and Young [76, §4.8] or Munkres [137, §80].

2.5.2 Local view

Let v be a node in an anonymous, port-numbered network $\mathcal{G}$. Let $\mathcal{T}$ be the unfolding of $\mathcal{G}$, and let v' be a preimage of v in the covering map f, as in the example of Figure 2.3.

The *radius-r local view* of the node v is the subgraph $\mathcal{T}(v', r)$ of $\mathcal{T}$ induced by $B_{\mathcal{T}}(v', r)$. Put otherwise, the radius-r local view of v is the radius-r neighbourhood of its preimage v' in the unfolding. See Figure 2.3 for an illustration in the case $r = 3$. The local view does not depend on the choice of $v' \in f^{-1}(\{v\})$.

Now we are ready to characterise exactly what we can do in the port numbering model. We begin with the good news. In a deterministic local algorithm with local horizon r, each node v can construct its radius-r local view [21, 174]. There is a simple local algorithm that gathers this information in r communication rounds: Initially, each node knows its radius-0 local view. On the communication round i, each node floods its radius-$(i-1)$ local view to each neighbour, and includes the outgoing port number in the message. After the communication round i, each node pieces together the local views received from its neighbour; this results in the radius-i local view. We can also gather the local input for each node in the local view. Hence, in a local algorithm each node v can choose its output o_v based on all information that is available in its radius-r local view. If the local views of nodes u and v differ, then the local outputs of the nodes u and v can differ as well.

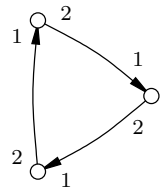

Figure 2.4: A communication graph with a port numbering and an orientation; cf. Figure 2.1.

The bad news is that choosing the local output based on the local view is, in a sense, the only thing that one can do in a local algorithm [6, 21, 174]. This is easily understood if we consider the covering map f from the unfolding $\mathcal{T}$ to the communication graph $\mathcal{G}$, and apply the same local algorithm $\mathcal{A}$ in both $\mathcal{T}$ and $\mathcal{G}$. Initially, for each node v' in $\mathcal{T}$, the local state of v' in $\mathcal{T}$ and $v = f(v')$ in $\mathcal{G}$ is the same. Furthermore, on each communication round, v' and v perform the same local computation, send the same messages, and receive the same messages – here we use the fact that f is a local isomorphism that preserves the port numbering. Hence, after r communication rounds, both v' and v must produce the same output. Therefore the output of v in a local algorithm with local horizon r only depends on its local view $B_{\mathcal{T}}(v', r)$.

Among others, this shows that a local algorithm cannot distinguish between $\mathcal{G}$ and $\mathcal{H}$ in Figure 2.3 because they have the same unfolding $\mathcal{T}$. The node v in $\mathcal{G}$, the nodes v_1 and v_2 in $\mathcal{H}$, and the node v' in $\mathcal{T}$ all produce the same output. We can see that a local algorithm in an anonymous network cannot even detect if there are triangles (3-cycles) in the network.

2.5.3 Graphs with orientation

So far we have assumed that we have a port numbering in the communication graph $\mathcal{G}$. We proceed to study a slightly stronger assumption [130]: in addition to the port numbering, we are given an *orientation* of the graph $\mathcal{G}$. That is, for each edge $\{u, v\} \in E$, we have chosen exactly one direction, either (u, v) or (v, u). See Figure 2.4 for an illustration. In a port numbering, each node chooses an ordering on incident edges. In an orientation, each *edge* chooses an ordering on incident *nodes*.

We use the standard terminology for directed graphs: If the edge $\{u, v\}$ has the orientation (u, v), then u is a *predecessor* of v and v is a *successor* of u. The *in-degree* of a node is the number of predecessors, i.e., the number of edges entering the node. Similarly, the *out-degree* of a node is the number of successors, i.e., the number of edges leaving the node.

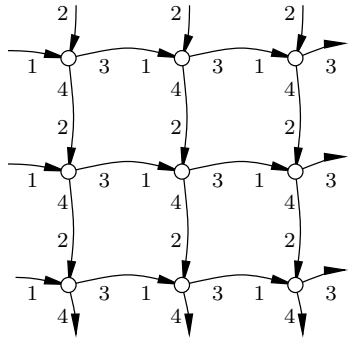

Figure 2.5: A 4-regular graph with a port numbering and an orientation.

At first sight, having an arbitrary orientation in addition to an arbitrary port numbering does not seem to help much. In an n-cycle, we can have all edges directed consistently in a clockwise direction, as shown in Figure 2.4. Hence we obtain the same negative results as in the case of an n-cycle with only a port numbering. More generally, we can construct a d-regular graph for any *even* constant d such that the local view of each node is identical [138], in spite of a port numbering and an orientation; see Figure 2.5 for an example. Furthermore, as shown in Figure 2.6, having an orientation does not help one to tell a graph from its cover.

Surprisingly, it turns out that in graphs where every node has an *odd* degree, an orientation together with a port numbering is enough to break the symmetry in the following sense: the output of a non-isolated node v is different from the output of at least one neighbour u of v.

Example 2.2. Consider a 3-regular graph with a port numbering and an orientation. Let v be an arbitrary node; we show that the local view of v is different from the local view of at least one neighbour of v. Figure 2.7 illustrates the three possible cases. In Figure 2.7a, the node v and its neighbour u have different out-degrees; hence the local view of v differs from the local view of u. Otherwise the out-degree of v and each neighbour of v is the same. The common out-degree of v and its neighbours is either 1 or 2. Figure 2.7b illustrates the case where the common out-degree is 2. In this case the local view of s is necessarily different from the local view of t: both have exactly one predecessor, and the port numbers assigned to these unique incoming edges are different because $p(v, s) \neq p(v, t)$. Therefore the local view of v is different from the local view of s or t (or both). Figure 2.7c illustrates the case where the common out-degree is 1; this is analogous to the case of the out-degree 2.

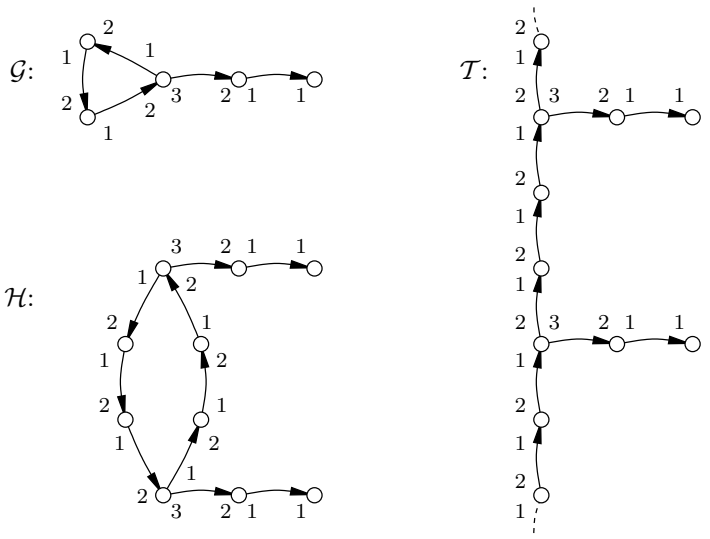

Figure 2.6: Covering graphs with a port numbering and an orientation; cf. Figure 2.3.

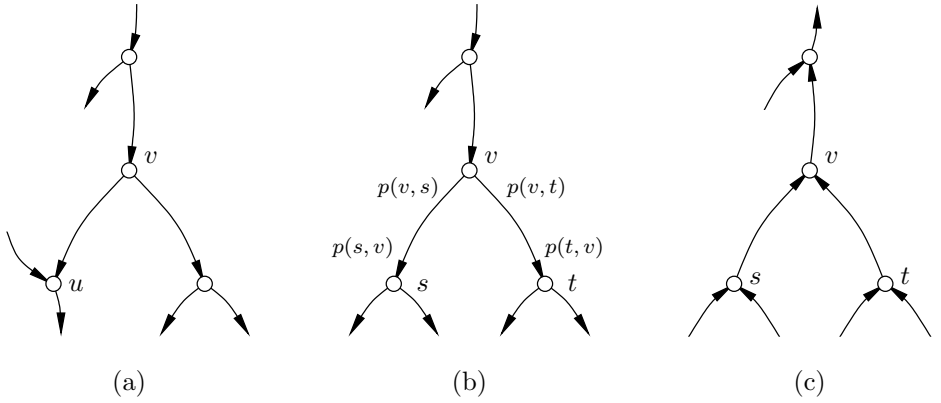

Figure 2.7: A 3-regular graph with a port numbering and an orientation. (a) Different out-degrees. (b) Out-degree is 2. (c) Out-degree is 1.

This argument can be generalised to any graph, as long as the degree of each node is odd. We present the details in Section 2.7.3 when we review a local algorithm for weak colouring [130, 138].

2.5.4 Graphs with unique identifiers

We can make an even stronger assumption: each node v has a *globally unique identifier* as part of its local input i_v. Usually the identifiers are assumed to be a permutation of $\{1, 2, \ldots, |V|\}$; see, for example, Linial [121]. Naturally we require that the local algorithm solves the problem for any permutation, not just for some subset of permutations.

Globally unique identifiers are a standard assumption in the field of local algorithms. This is a strictly stronger assumption than having only port numbers and an orientation available; all negative results for the case of globally unique identifiers imply negative results in anonymous networks and anonymous oriented networks.

Unfortunately, the assumption on globally unique identifiers means that the problem is not of distributed constant size: the number of bits required to encode the identifiers increases as the size of the network increases. In practice, for many algorithms that are designed under the assumption of globally unique identifiers, it is sufficient to have *locally unique identifiers*. That is, we assume that a local algorithm with local horizon r has access to identifiers that are unique within every radius-r neighbourhood in the communication graph $\mathcal{G}$. In a bounded-degree graph, it is then possible to choose identifiers that are locally unique but have constant size.

With globally or locally unique identifiers, a node v in the graph $\mathcal{G}$ can tell for each pair of nodes s, t in its radius-r local view whether $f(s) = f(t)$, that is, whether they represent the same node in the original graph $\mathcal{G}$. This implies that each node v can reconstruct the subgraph $\mathcal{G}[v, r]$ of $\mathcal{G}$. See Figure 2.8 for an illustration. Hence, when we study local algorithms in networks with unique identifiers, it is sufficient to present a function that maps the subgraph $\mathcal{G}[v, r]$ to the local output o_v [138].

We note that the difference between $\mathcal{G}(v, r)$ and $\mathcal{G}[v, r]$ is usually immaterial. After all, $\mathcal{G}[v, r + 1]$ contains $\mathcal{G}(v, r)$ as a subgraph, and $\mathcal{G}(v, r)$ contains $\mathcal{G}[v, r]$ as a subgraph. We are typically not interested in additive constants in the local horizon r. Hence we can use either $\mathcal{G}(v, r)$ or $\mathcal{G}[v, r]$ to derive both positive and negative results, whichever is more convenient. If the local output cannot be determined based on $\mathcal{G}(v, r)$ for any constant r, then there is no local algorithm for the task; if it can be determined based on $\mathcal{G}(v, r)$ for some constant r, then there is a local algorithm. We

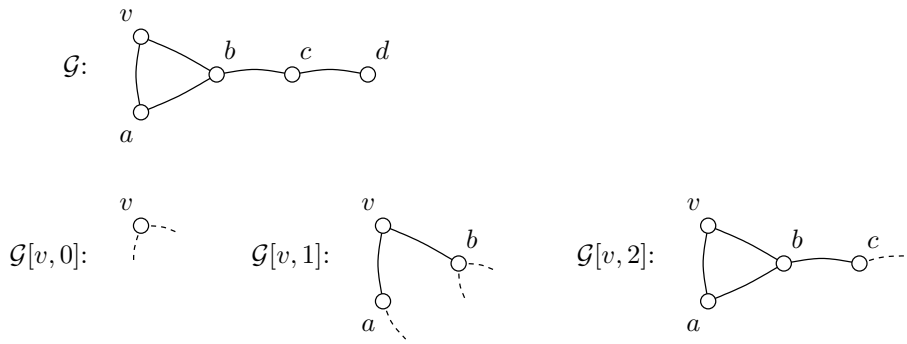

Figure 2.8: The local view of the node v in a graph $\mathcal{G}$ with unique node identifiers.

can even go as far as to use this as the *definition* of a local algorithm, if we study networks with unique identifiers.

2.6 Negative results

In this section we review negative results for local algorithms. There are two simple arguments that can be used to show that a problem cannot have a local algorithm: *inherently non-local problems* and the *impossibility of symmetry-breaking*.

A problem is inherently non-local if the output at a node u may depend on the input at a node v with $d_{\mathcal{G}}(u,v) = \Omega(|V|)$. By definition, a local algorithm cannot solve a problem that is inherently non-local. Constructing a spanning tree is a classical example of a simple problem that is inherently non-local; see Figure 2.9 for an illustration. Finding a stable matching is another example of a non-local problem [55].

A problem such as finding a maximal matching is, in a sense, much more local. For example, if we already have a solution M for a graph $\mathcal{G}$, a local change in $\mathcal{G}$ requires only local changes in the solution M. However, as we discussed in Section 2.5, it is not possible to break the symmetry with a local algorithm in an n-cycle if the nodes are anonymous; a port numbering and an orientation of $\mathcal{G}$ do not help. Therefore it is not possible to find a maximal matching in an anonymous n-cycle, oriented or not. The same negative results apply to the problems of finding a maximal independent set, a vertex colouring, an edge colouring, a weak colouring, an $O(1)$-approximation of a maximum matching, an $O(1)$-approximation of a maximum independent set, a $(3-\varepsilon)$-approximation of a minimum

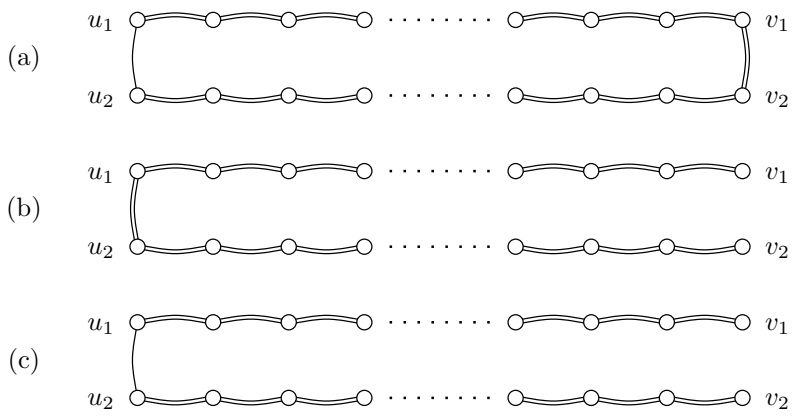

Figure 2.9: (a) An n-cycle $\mathcal{G}$; the double lines show a spanning tree. (b) A local change in the graph $\mathcal{G}$ near the nodes v_1 and v_2 requires a non-local change in the spanning tree near the nodes u_1 and u_2. (c) A local algorithm cannot even verify whether a given set of edges is a spanning tree or not [96]. In every local neighbourhood, this non-tree looks similar to a spanning tree in part (a) or (b).

dominating set, and a $(2-\varepsilon)$-approximation of a minimum vertex cover in anonymous n-cycles. As a straightforward generalisation, there is no local $o(\Delta)$-approximation algorithm for the minimum dominating set problem in anonymous bounded-degree graphs.

In this section, we focus on negative results that hold even if globally unique identifiers are available. The negative results are summarised in Tables 2.1 and 2.2.

2.6.1 Comparable identifiers

Let us first focus on *order-invariant* algorithms: we assume that unique node identifiers are available, but the algorithm is only allowed to compare the identifiers and not access their numerical value.

It turns out that being able to compare identifiers does not help much in symmetry breaking. For example, in an n-cycle, we can assign the node identifiers $1, 2, \ldots, n$ in an increasing order. If we pick any two nodes $u, v \in U = \{r+1, r+2, \ldots, n-r\}$, then the radius-$r$ neighbourhood of u looks identical to the radius-r neighbourhood of v, assuming that a node can only exploit the ordering of the identifiers. Therefore every node in U must make the same decision; see, e.g., Kuhn [102, §2.7.2]. We immediately obtain the same negative results that we had for anonymous networks in an n-cycle: for example, there is no local algorithm for finding a maximal matching, a

Problem	Graph family	References
Maximal independent set	cycle	[121]
Maximal matching	cycle	[121] cor.
Vertex 3-colouring	cycle	[121]
Vertex Δ-colouring	$(\Delta+1)$-coloured tree	[102]
Edge colouring	cycle	[121] cor.
Weak colouring	$2k$-regular	[138]

cor. = corollary, see text

Table 2.1: Problems that cannot be solved with a deterministic local algorithm, even if there are unique node identifiers. The problems are defined in Section 2.4.

maximal independent set, a vertex colouring, an edge colouring, or a weak colouring.

2.6.2 Numerical identifiers

A natural approach would be to exploit the numerical values of the identifiers; after all, this is exactly what the classical (distributed but not constant-time) algorithm for vertex colouring by Cole and Vishkin [35] does.

Unfortunately, a general result by Naor and Stockmeyer [138] shows that local algorithms for so-called locally checkable labellings – these include vertex colourings and maximal independent sets in bounded-degree graphs – do not benefit from the numerical values of the identifiers: if there is a local algorithm that uses the numerical values, there is an order-invariant local algorithm as well.

More specifically, Linial [121] shows that a synchronous distributed algorithm for vertex 3-colouring in an n-cycle with unique identifiers requires $\Omega(\log^* n)$ communication rounds. Here $\log^* n$ denotes the iterated (base-2) logarithm of n, that is, the smallest integer $k \geq 0$ such that k iterated applications of the function $x \mapsto \log_2 x$ to the initial value n results in a value at most 1.

Linial's result holds even under the assumption that there is a consistent clockwise orientation in the n-cycle. As a direct implication, an algorithm for finding an edge 3-colouring, a maximal independent set, or a maximal matching in an n-cycle requires $\Omega(\log^* n)$ communication rounds as well. The barrier of $\Omega(\log^* n)$ is hard to break even if we are allowed to provide arbitrary instance-specific *advice* to some nodes in the network [56].

Problem	Approx. factor	Graph family	References
Independent set	$O(1)$	cycle	[39, 119]
Matching	$O(1)$	general	[102, 108, 134]
	$O(1)$	cycle	[39]
Edge cover	$2-\varepsilon$	cycle	[39, 119] cor.
Vertex cover	$O(1)$	general	[102, 105, 134]
	$2-\varepsilon$	cycle	[39, 119] cor.
Dominating set	$O(1)$	general	[102, 105, 134]
	$O(1)$	unit-disk	[119]
	$2k+1-\varepsilon$	$2k$-regular	[39] cor.
	$k+1-\varepsilon$	$(2k{+}1)$-regular, 2-C	[10]
	$5-\varepsilon$	4-regular, planar	[39]
	$3-\varepsilon$	cycle	[39, 119]
Domatic partition	$3-\varepsilon$	cycle	[39, 119] cor.
Edge dominating set	$3-\varepsilon$	cycle	[39, 119] cor.
Maximum cut	$O(1)$	cycle	[39, 119] cor.
0/1 packing LP	$O(1)$	general	[102, 108, 134]
0/1 covering LP	$O(1)$	general	[102, 105, 134]
0/1 max-min LP	α	bounded-degree	Chapter 3

$\varepsilon > 0$, $\alpha = \Delta_I(1 - 1/\Delta_K)$
2-C = bicoloured graphs, i.e., a 2-colouring is given
cor. = corollary, see text

Table 2.2: Approximation factors that cannot be achieved with a deterministic local algorithm, even if there are unique node identifiers.

2.6.3 Approximations for combinatorial problems

So far we have seen that there are no local algorithms for problems such as vertex colouring, edge colouring, maximal independent set, or maximal matching. However, it is possible to find a feasible independent set or a matching with a local algorithm (the empty set), and similarly there is a trivial local algorithm for finding a vertex cover or a dominating set (the set of all nodes). This raises the question of whether there is a local approximation algorithm for any of these problems, with a nontrivial approximation guarantee.

Unfortunately, this does not seem to be the case. Czygrinow et al. [39] and Lenzen and Wattenhofer [119] show that it is not possible to find a constant-factor approximation of a maximum independent set or a max-

imum matching in an n-cycle with a deterministic local algorithm. Czygrinow et al.'s elegant proof uses Ramsey's theorem [65, 148]; Lenzen and Wattenhofer build on Linial's [121] work.

These results imply that there is no local constant-factor approximation algorithm for the maximum cut problem in an n-cycle. If we can find a cut $\{X, Y\}$ of size k in an n-cycle, then it is possible to find a matching with at least $k/3$ edges as well. The edges that cross the cut form a bipartite graph $\mathcal{H}$, the cut $\{X, Y\}$ is a 2-colouring of the bipartite graph $\mathcal{H}$, and the algorithm that we will describe in Section 2.7.1 finds a maximal matching in the 2-coloured graph $\mathcal{H}$.

As another corollary, there is no local $(2-\varepsilon)$-approximation algorithm for the edge cover problem in a cycle. To see this, consider a $2n$-cycle $\mathcal{G} = (V, E)$. A minimum edge cover has n edges, and a maximum matching has n edges as well. Hence a $(2-\varepsilon)$-approximate edge cover $C \subseteq E$ has at most $(2-\varepsilon)n$ edges, and its complement $M = E \setminus C$ has at least εn edges. Furthermore, each $v \in V$ is covered by at least one edge in C; therefore each $v \in V$ is covered by at most one edge in M. Hence M is a matching, and within factor $1/\varepsilon$ of the optimum.

An analogous argument shows that there is no local $(2-\varepsilon)$-approximation algorithm for the vertex cover problem in an n-cycle. Furthermore, there is no local $(3-\varepsilon)$-approximation algorithm for the dominating set problem. Note that the complement $X = V \setminus D$ of a dominating set D in an n-cycle can be turned into an independent set [39]: a node $v \in X$ is adjacent to at most one other node $u \in X \setminus \{v\}$. Exchanging the roles of edges and nodes, the same argument shows that there is no local $(3-\varepsilon)$-approximation algorithm for the edge dominating set problem in a cycle. Moreover, we cannot find more than one disjoint dominating set in a cycle; because a $3n$-cycle has 3 disjoint dominating sets, this shows that no local algorithm can find a $(3-\varepsilon)$-approximation of a maximum domatic partition.

More generally, Czygrinow et al. [39] and Lenzen and Wattenhofer [119] show that for any constant $\varepsilon > 0$, a local algorithm cannot produce a factor $2k+1-\varepsilon$ approximation of a minimum dominating set in $2k$-regular graphs. The basic argument is as follows. Figure 2.10a shows a $(2k+1)n$-cycle $\mathcal{G} = (V, E)$. Using a local algorithm, we can construct a $2k$-regular graph $\mathcal{H} = (V, E')$, as illustrated in Figure 2.10b. A minimum dominating set of $\mathcal{H}$ has n nodes (Figure 2.10c); therefore a hypothetical $(2k+1-\varepsilon)$-approximation algorithm has to return a dominating set D with at most $(2k+1-\varepsilon)n$ nodes (Figure 2.10d). Therefore its complement $X = V \setminus D$ has at least εn nodes. Furthermore, because D is a dominating set, there is no path with more than $2k$ nodes in the subgraph of $\mathcal{G}$ induced by X (Figure 2.10e). Hence we can construct an independent set I with at

least $\varepsilon n/(2k)$ nodes (Figure 2.10f), which is a contradiction with the local inapproximability of the independent set problem in cycles.

Czygrinow et al. [39] consider the case $k = 2$ to show that a local algorithm cannot find a factor $5 - \varepsilon$ approximation of a minimum dominating set in planar graphs. Lenzen and Wattenhofer [119] consider a general k to show that a local algorithm cannot find a constant-factor approximation of a minimum dominating set in unit-disk graphs (see Section 2.9.1 for the definition).

In the above proof, we have focused on regular graphs with an even degree. As we saw in Section 2.5.3, some amount of symmetry-breaking is possible in graphs where each node has an odd degree. Nevertheless, we can derive a slightly weaker result for regular graphs with an odd degree: a local algorithm cannot produce a factor $k + 1 - \varepsilon$ approximation of a minimum dominating set in $(2k + 1)$-regular graphs [10]. To see this, consider a $(k + 1)n$-cycle $\mathcal{G} = (V, E)$; see Figure 2.11a. We can use a local algorithm to construct a $(2k+1)$-regular graph $\mathcal{H}$ as illustrated in Figure 2.11b; each original node $v \in V$ simulates a pair of nodes, v' and v'', in $\mathcal{H}$. A minimum dominating set of $\mathcal{H}$ has n nodes (Figure 2.11c). A hypothetical $(k + 1 - \varepsilon)$-approximation algorithm has to return a dominating set D with at most $(k + 1 - \varepsilon)n$ nodes (Figure 2.11d). Let $X \subseteq V$ consist of the nodes $v \in V$ such that both $v' \notin D$ and $v'' \notin D$ (Figure 2.11e). We know that the size of X is at least εn; furthermore, X does not induce paths with more than $2k$ nodes in $\mathcal{G}$. Hence we can construct an independent set I with at least $\varepsilon n/(2k)$ nodes (Figure 2.11f), a contradiction.

This negative result holds even in bipartite graphs where a 2-colouring is given as part of the local input. Incidentally, the construction in Figure 2.11 is the so-called bicoloured double cover of the construction in Figure 2.10; we will revisit bicoloured double covers in more detail when we present positive results in Section 2.7.1.

2.6.4 Approximations for LPs

The negative results in the previous section build on the impossibility of symmetry breaking in combinatorial problems. However, there are also negative results for linear programs.

Bartal et al. [19] observe that a $(1 + \varepsilon)$-approximation algorithm for packing and covering LPs requires $\Omega(1/\varepsilon)$ communication rounds.

Kuhn, Moscibroda, and Wattenhofer [102, 103, 105, 108, 134] show that it is not possible to find a constant-factor approximation of a minimum vertex cover, minimum dominating set, or maximum matching in general graphs with a local algorithm, if there is no degree bound. The results

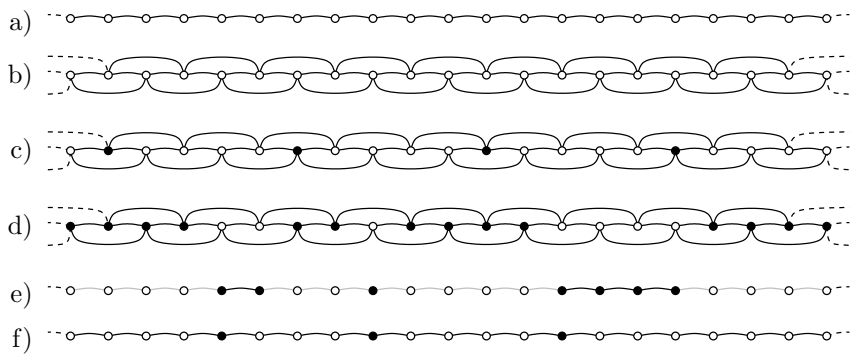

Figure 2.10: There is no local $(2k+1-\varepsilon)$-approximation algorithm for the dominating set problem in $2k$-regular graphs (the case $k = 2$).

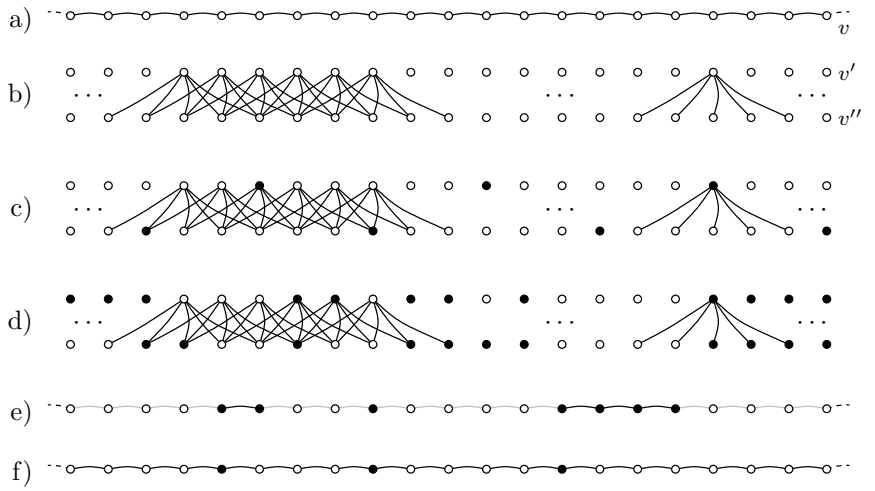

Figure 2.11: There is no local $(k+1-\varepsilon)$-approximation algorithm for the dominating set problem in $(2k+1)$-regular graphs (the case $k = 2$).

Problem	Graph family	Model	References
Maximal matching	bicoloured, B-D	P	[70]
ε-stable matching	bicoloured, B-D	P	[55]
Vertex (Δ+1)-colouring	k-coloured, B-D	P	[35, 64, 102, 111, 121]
Weak colouring	odd degree, B-D	P+O	[130, 138]

B-D = bounded-degree graph
P = algorithm uses only a port numbering
P+O = algorithm uses only a port numbering and an orientation

Table 2.3: Deterministic local algorithms. The problems are defined in Section 2.4.

extend to the LP relaxations of these problems as well, and hence to 0/1 packing LPs and 0/1 covering LPs.

Our work in Papers I and II presents tight lower bounds for the local approximability of max-min LPs; see Chapter 3 for details.

2.7 Positive results

In spite of all negative results that we saw in Section 2.6, a few local algorithms are known. In this section, we review known deterministic local algorithms; prominent positive results are also summarised in Tables 2.3 and 2.4.

We begin with two different techniques, both of which yield a local approximation algorithm for the vertex cover problem: Section 2.7.1 presents a technique based on bicoloured covering graphs; Section 2.7.2 presents a linear programming approach.

2.7.1 Bicoloured matching and vertex cover

In a centralised setting, there is a simple 2-approximation algorithm for the vertex cover problem: find any maximal matching and take the endpoints.[1] We cannot find a maximal matching with a local algorithm in general graphs; nevertheless, we can apply the same basic idea indirectly to design a local approximation algorithm for the vertex cover problem.

We say that the communication graph $\mathcal{G}$ is *bicoloured* if a vertex 2-colouring of $\mathcal{G}$ is given as part of the local input – each node knows whether

[1] Papadimitriou and Steiglitz [140] attribute this algorithm to Fanica Gavril and Mihalis Yannakakis.

Problem	Approx. factor		Graph family	Model	References
Matching	$1+\varepsilon$	$*$	2-C, B-D	P	[10, 70]
	$(\Delta+1)/2$	$*$	W-C, B-D	P	[10]
Weighted matching	$2+\varepsilon$		2-C, B-D	P	[55]
Simple 2-matching	$2+\varepsilon$		B-D	P	[10, 147] cor.
Edge cover	2	$*$	general	P	
Vertex cover	2	$*$	regular	P	
	6		unit-disk	P	[102, 169]
	$4+\varepsilon$		B-D		[134]
	3		B-D	P	[147]
	$2+\varepsilon$		B-D		[102, 108]
	2		B-D	P	[9]
Dominating set	$\Delta+1$		B-D	P	
	$2\lfloor\Delta/2\rfloor+1$	$*$	B-D	P+O	[10]
	$(\Delta+1)/2$	$*$	W-C, B-D	P	[10]
	$O(1)$		planar		[39]
	74		planar		[118]
Domatic partition	$(\delta+1)/2$		W-C, B-D	P	
Edge domin. set	4		B-D	P	[8]
Maximum cut	Δ		W-C, B-D	P	
Set cover	Δ_V		B-D	P	
	$\Delta_K+\varepsilon$		B-D		[102, 108]
Packing LP	Δ_I		B-D	P	[143]
0/1 packing LP	$1+\varepsilon$	$*$	B-D		[102, 108]
0/1 covering LP	$1+\varepsilon$	$*$	B-D		[102, 108]
Max-min LP	Δ_I		B-D	P	[143] cor.
	$\alpha+\varepsilon$	$*$	B-D	P	Chapter 3

$\varepsilon > 0$, $\alpha = \Delta_I(1-1/\Delta_K)$, δ = minimum degree of $\mathcal{G}$
$*$ = tight approximation ratio (matching negative result)
B-D = bounded-degree graph
2-C = bicoloured graphs, i.e., a 2-colouring is given
W-C = a weak 2-colouring is given or can be found locally [130, 138]
P = algorithm uses only a port numbering
P+O = algorithm uses only a port numbering and an orientation
cor. = corollary, see text

Table 2.4: Deterministic local approximation algorithms. The algorithms without references are trivial; see text.

it is *black* or *white*. Clearly the graph $\mathcal{G}$ has to be bipartite; otherwise there is no 2-colouring.

Hańćkowiak et al. [70] present a simple local algorithm for the problem of finding a maximal matching in a bicoloured bounded-degree graph. A port numbering is enough; unique node identifiers are not needed. The algorithm performs the following two steps repeatedly:

1. Each unmatched black node sends a proposal to one of its white neighbours. The neighbours are chosen in the order of port numbers.

2. Each white node accepts the first proposal that it receives. If several proposals are received in the same round, ties are broken with port numbers.

After 2Δ steps, this results in a maximal matching M. To see that M is maximal, consider an edge $e = \{u, v\} \in E \setminus M$ such that u is a black node and v is a white node. One of the following holds: (i) u never sent a proposal to v, or (ii) v rejected the proposal from u. In the case (i), the node u is matched, and in the case (ii), the node v is matched. Hence $M \cup \{e\}$ is not a matching.

Now we know how to find a maximal matching in a bicoloured graph. It turns out that with the help of this simple algorithm, it is possible to find a 3-approximation of a minimum vertex cover in an arbitrary bounded-degree graph [147]. The key observation is the following: for any graph $\mathcal{G}$, we can construct the bicoloured graph $\mathcal{H}$ that is the *bipartite double cover* [6, 23, 83] of $\mathcal{G}$; see Figure 2.12 for an illustration.

The bipartite double cover of $\mathcal{G}$, also known as the Kronecker double cover, is the Kronecker product [166] of the graphs $\mathcal{G}$ and K_2. In essence, for each original node v in the graph $\mathcal{G}$, we create two copies: a black copy and a white copy. If u and v are adjacent in the original graph $\mathcal{G}$, then the black copy of u is adjacent to the white copy of v in the graph $\mathcal{H}$ and vice versa. Port numbers can be inherited from $\mathcal{G}$. It follows that $\mathcal{H}$ is a covering graph of $\mathcal{G}$ (see Section 2.5.1). Furthermore, it is a *double* cover: the covering map f maps exactly 2 nodes of $\mathcal{H}$ onto each node of $\mathcal{G}$.

Now let $\mathcal{A}$ be the local algorithm for maximal matchings in bicoloured graphs. A local algorithm that runs in a general graph $\mathcal{G}$ can *simulate* the behaviour of $\mathcal{A}$ in the bicoloured double cover $\mathcal{H}$; a node v in $\mathcal{G}$ is responsible for simulating the behaviour of both its black copy and its white copy. Once the simulation completes, each node can inspect the output produced by its two copies.

Hence we can find a maximal matching M in the bicoloured double cover $\mathcal{H}$ and map it back to the original graph $\mathcal{G}$. This gives us a subset of

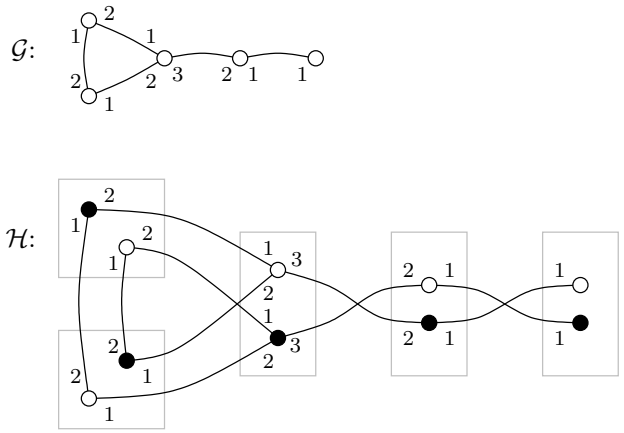

Figure 2.12: The bicoloured graph $\mathcal{H}$ is the bipartite double cover of the graph $\mathcal{G}$. Note that the graphs are isomorphic to those in Figure 2.3.

edges $X \subseteq E$ in $\mathcal{G} = (V, E)$ with the following properties: (i) for each node $v \in V$, there are at most 2 edges incident to v in X; and (ii) for each edge $\{u, v\} \in E$, at least one of u, v is incident to an edge $x \in X$. Put otherwise, X is a simple 2-matching and its endpoints are a vertex cover $C \subseteq V$.

A minimum-size vertex cover C^* must cover all edges, including the edges in the simple 2-matching X. A node $v \in C^*$ can cover at most 2 edges in X, and an edge in X is covered by at most 2 nodes in C. Hence $|C| \leq 2|X| \leq 4|C^*|$. We have a simple algorithm that finds a 4-approximation of a minimum vertex cover; the algorithm is local and it does not need unique node identifiers.

A more careful analysis shows that $|C| \leq 3|C^*|$, so this is actually a local 3-approximation algorithm for the vertex cover problem [147]; the bottleneck is a path of length 2 in the set X. A repeated application of bicoloured double covers can be used to find a 2-approximation of a minimum vertex cover [9]. See Hańćkowiak et al. [71] for an example of a non-local distributed algorithm that exploits bicoloured double covers.

2.7.2 Linear programs and vertex cover

In a centralised setting, another 2-approximation algorithm for the vertex cover problem can be obtained by deterministic LP rounding [74]: (i) solve the LP relaxation of the vertex cover problem; and (ii) output the set of nodes $v \in V$ with $x_v \geq 1/2$. Unlike the algorithm based on a maximal matching, this directly generalises to the problem of finding a minimum-weight vertex cover as well.

To obtain the LP relaxation of a vertex cover instance, we first write the vertex cover instance as a set cover instance. The set cover instance determines an integer program of the form (2.1). The LP relaxation of the vertex cover instance is the corresponding 0/1 covering LP (2.2).

A local algorithm cannot find an exact solution of a linear program; the problem is inherently non-local. However, local approximation algorithms for packing and covering LPs are known. The first such algorithm was presented by Papadimitriou and Yannakakis [143]. In this simple algorithm, the "capacity" of each constraint in a packing LP is distributed evenly among the adjacent agents; the approximation factor is Δ_I (see Section 3.4.1 for details). Kuhn and Wattenhofer [110] present improved local approximation algorithm for special cases of packing and covering LPs. Finally, Kuhn et al. [102, 103, 108] present local approximation algorithms for general packing LPs and covering LPs. Among others, they show that 0/1 packing LPs and 0/1 covering LPs admit local approximation schemes in bounded-degree graphs.

Hence we can find a factor $1+\varepsilon$ approximation of the LP relaxation of the minimum vertex cover problem in bounded-degree graphs. Therefore deterministic LP rounding yields a factor $2+\varepsilon$ approximation of a minimum vertex cover in bounded-degree graphs, for any $\varepsilon > 0$. Note that the vertex cover problem is a special case of the set cover problem with $\Delta_K = 2$; the same technique of deterministic LP rounding can be applied to design a local $(\Delta_K + \varepsilon)$-approximation algorithm for the set cover problem in bounded-degree graphs, for an arbitrary Δ_K.

It is also possible to use the primal-dual schema [140, 162] to design an algorithm that finds an approximation of a minimum vertex cover directly without a rounding step. Moscibroda [134, §6.1] uses this approach to design a $(4+\varepsilon)$-approximation algorithm for the vertex cover problem in bounded-degree graphs.

2.7.3 Weak colouring

Weak colouring provided the first example of a nontrivial combinatorial problem that admits a local algorithm. Naor and Stockmeyer [138] show that if $\mathcal{G}$ is a bounded-degree graph and every node of $\mathcal{G}$ has an *odd* degree, then there is a local algorithm that finds a weak 2-colouring of $\mathcal{G}$. Mayer et al. [130] further show that this is possible without unique identifiers; a port numbering and an orientation is enough.

Let us now show how to find a weak colouring with $\Delta^{O(\Delta)}$ colours, using only a port numbering and an orientation. Each node v is coloured with a

vector $\mathbf{x}(v)$ that consists of the following components:

- the out-degree and in-degree of v
- $p(u, v)$ for each successor u of v, in the order of increasing $p(v, u)$
- $p(u, v)$ for each predecessor u of v, in the order of increasing $p(v, u)$.

Example 2.3. Assume that v has three neighbours, u_1, u_2, u_3, with the port numbers $p(v, u_1) = 1$, $p(v, u_2) = 2$, and $p(v, u_3) = 3$. Assume that the edges are oriented (v, u_1), (u_2, v), and (v, u_3). Then

$$\mathbf{x}(v) = \big(2,\ 1,\ p(u_1, v),\ p(u_3, v),\ p(u_2, v)\big),$$

that is, the out-degree, the in-degree, two port numbers for the edges leaving v, and one port number for the edges entering v. Note that the elements are port numbers of the form $p(\cdot, v)$ but they are ordered by the port numbers $p(v, \cdot)$.

To see that the vectors $\mathbf{x}(v)$ are a weak colouring of the graph, we generalise the argument that we saw in Section 2.5.3. Let v be an arbitrary node. We need to show that there is a neighbour u of v such that $\mathbf{x}(u) \neq \mathbf{x}(v)$. If the first two components (out-degree and in-degree) are equal in $\mathbf{x}(v)$ and $\mathbf{x}(u)$ for each neighbour u of v, we can consider the following two cases.

First, assume that the in-degree of v is k and the out-degree of v is at least $k + 1$. Then there are at least $k + 1$ successors of v, and each successor has k predecessors. By the pigeonhole principle, there are at least two successors of v, call them s and t, and an index i with the following property: the element i of the vector $\mathbf{x}(s)$ is $p(v, s)$ and the element i of the vector $\mathbf{x}(t)$ is $p(v, t)$. Therefore $\mathbf{x}(s) \neq \mathbf{x}(t)$, and we cannot have $\mathbf{x}(v) = \mathbf{x}(s) = \mathbf{x}(t)$. Put otherwise, v has a different colour from s or t (or both).

Second, assume that the in-degree of v is at least $k + 1$ and the out-degree of v is k. This case is analogous: we apply the pigeonhole principle to the predecessors of v.

Note that this analysis does not go through in a graph with an even degree. We may have in-degrees equal to out-degrees, and therefore we cannot invoke the pigeonhole principle – consider, for example, the 4-regular graph in Figure 2.5 on page 20.

So far we have seen how to find a weak colouring with a constant but large number of colours. In the following section, we review techniques that can be used to reduce the number of colours.

2.7.4 Colour reduction

A local algorithm cannot find a vertex colouring, but it can *decrease* the number of colours. Given a k-colouring for a constant k, it is easy to design a local algorithm that finds a $(\Delta+1)$-colouring. In essence, we run a greedy algorithm. The original k-colouring partitions the network in k layers. In the step $i = 1, 2, \ldots, k$, the nodes on the layer i choose a colour that is not used by any of their neighbours on the layers $1, 2, \ldots, i-1$. The nodes on each layer form an independent set; hence they can make their choices independently in parallel. In the worst case, $\Delta + 1$ colours are needed.

A much more efficient algorithm can be designed by exploiting the numerical values of the original colours. In a cycle, the technique originally presented by Cole and Vishkin [35] allows one to decrease the number of colours from k to $O(\log k)$ in one step. Iterating the procedure, we can turn a k-colouring into a 3-colouring with a local algorithm in $O(\log^* k)$ steps. The textbook by Cormen et al. [36, §30.5] has a good illustration of the Cole–Vishkin technique; in essence, a node with a b-bit label relabels itself with an $O(\log b)$-bit label (i, x) that identifies the index i and the value x of the first bit in its old label that differs from the next node in the cycle.

The same basic idea can be applied in general bounded-degree graphs as well [64]. If k is a constant, a k-colouring can be turned into a $(\Delta + 1)$-colouring with a local algorithm in $O(\Delta \log \Delta + \log^* k)$ steps [102, 111]; see also Attiya et al.'s [11] algorithm that finds a $(\Delta + 1)$-colouring assuming that a so-called t-orientation is given. Naor and Stockmeyer [138] show how to turn a weak k-colouring into a weak 2-colouring.

Naturally, if we are given a k-colouring for a constant k, we can also solve a number of other problems with a local algorithm. The symmetry has been broken and, for example, finding a maximal independent set is then easy [11, 14]. The following algorithm provides a reasonable trade-off between efficiency and simplicity: first apply a colour reduction algorithm, and then find a maximal independent set with a greedy algorithm. However, if Δ is large, more efficient alternatives exist; see, for example, the techniques used by Schneider and Wattenhofer [154].

2.7.5 Matching

As we have seen in Section 2.6.3, there is no local algorithm for finding a constant-factor approximation of a maximum matching in any family of graphs that contains n-cycles. However, positive results are known for bicoloured graphs and for graphs where each node has an odd degree.

We have already seen the algorithm by Hańćkowiak et al. [70] for finding a maximal matching in bicoloured bounded-degree graphs. A maximal matching is a 2-approximation of a maximum matching. It is possible to improve the approximation factor to $1 + \varepsilon$ for any $\varepsilon > 0$; it is enough to make sure that there is no augmenting path of length $O(1/\varepsilon)$ [10]. With the help of the bicoloured double cover from Section 2.7.1, this yields a $(2+\varepsilon)$-approximation of a maximum-size simple 2-matching in general bounded-degree graphs.

Bounded-degree graphs where each node has an odd degree admit a local Δ-approximation algorithm for the maximum matching problem; the algorithm proceeds as follows. We can first find a weak 2-colouring $c\colon V \to \{1, 2\}$ with the algorithm by Naor and Stockmeyer [138]. Let $X \subseteq E$ consist of the edges $\{u, v\} \in E$ with $c(u) \neq c(v)$. Let $\mathcal{H} = (U, X)$ be the subgraph of $\mathcal{G}$ induced by the edges in X, that is, U consists of the endpoints of the edges in X. Now $\mathcal{H}$ together with c is a bicoloured graph; therefore we can find a maximal matching M in $\mathcal{H}$; this is also a matching in $\mathcal{G}$. Let us now establish the approximation ratio. Let M^* be a maximum matching. Let $U \subseteq V$ consist of the nodes matched in M and let $U^* \subseteq V$ consists of the nodes matched in M^*. If $v \in U^* \setminus U$, then v is non-isolated in $\mathcal{G}$ and hence it is adjacent to a node u with the opposite colour, by the definition of a weak 2-colouring. Therefore $\{u, v\}$ is an edge in the subgraph $\mathcal{H}$, and since M is maximal, we have $u \in U$. Furthermore, u is adjacent to at least one node in U; therefore u is adjacent to at most $\Delta - 1$ nodes in $U^* \setminus U$. Summing over all nodes in U, we have $|U^* \setminus U| \leq (\Delta - 1)|U|$, that is, $|U^*| \leq \Delta|U|$ and $|M^*| \leq \Delta|M|$. The approximation factor can be further improved to $(\Delta + 1)/2$, which is tight [10].

The problem of finding a stable matching is inherently non-local, even in bicoloured graphs. However, it is possible to find an ε-stable matching in a bicoloured bounded-degree graph with a local algorithm [55]. In essence, the local algorithm runs the Gale–Shapley algorithm [59] for a constant number of rounds. The same local algorithm finds a $(2+\varepsilon)$-approximation of a maximum-weight matching in bicoloured bounded-degree graphs.

2.7.6 Domination

In a bounded-degree graph, the set of all nodes is a factor $\Delta + 1$ approximation of a minimum dominating set. If each node has an odd degree, it is possible to find a factor Δ approximation with a local algorithm, using a port numbering and an orientation: find a weak 2-colouring [130, 138] and output all nodes of colour 1 and all isolated nodes. This set is, by

definition, a dominating set: a non-isolated node of colour 2 is adjacent to at least one node of colour 1. To establish the approximation ratio, assume that $\mathcal{G} = (V, E)$ is connected; otherwise we can apply the result to each connected component. If $|V| = 1$, the claim is trivial. Otherwise, let D^* be an optimal dominating set in $\mathcal{G}$, let D_1 consist of the nodes of colour 1, and let $D_2 = V \setminus D_1$ consist of the nodes of colour 2. Now D^*, D_1, and D_2 are dominating sets in $\mathcal{G}$. Furthermore, for any dominating set D, it holds that $|D| \geq |V|/(\Delta + 1)$ because a node in D can only dominate at most Δ nodes outside D. Hence

$$|D_1| = |V| - |D_2| \leq |V| - \frac{|V|}{\Delta + 1} = \frac{\Delta |V|}{\Delta + 1} \leq \Delta |D^*|,$$

that is, D_1 is a Δ-approximation of a minimum dominating set in $\mathcal{G}$. Moreover, (D_1, D_2) is a domatic partition of size 2 if there is no isolated node, and a maximum domatic partition has at most $\delta + 1$ disjoint dominating sets where δ is the minimum degree of $\mathcal{G}$. Hence a weak 2-colouring provides a factor $(\delta + 1)/2$ approximation of a maximum domatic partition if there is no isolated node; the trivial solution (V) is optimal if there is an isolated node (i.e., if $\delta = 0$).

It is possible to perform local modifications in a weak 2-colouring so that the number of colour-1 nodes is at most as large as the number of colour-2 nodes [10]. This approach provides a factor $(\Delta + 1)/2$ approximation algorithm for the dominating set problem in graphs of odd degree, and a Δ-approximation in general graphs for an odd Δ.

Both Czygrinow et al. [39] and Lenzen and Wattenhofer [118] present a local, constant-factor approximation algorithm for the dominating set problem in planar graphs. The current best known approximation factor is 74, see Lenzen and Wattenhofer [118].

In Section 2.7.1 we saw how to use a maximal matching in a bicoloured double cover to find a constant-factor approximation for the vertex cover problem. The same technique provides a 4-approximation of a minimum edge dominating set as well [8].

2.7.7 Trivial algorithms

For some families of graphs, there is a trivial local approximation algorithm for the vertex cover problem. For example, the set of all nodes is a 2-approximation of a minimum vertex cover in regular graphs, and the set of all non-isolated nodes is a 6-approximation of a minimum vertex cover in unit-disk graphs [102, 169].

There is a trivial, local approximation algorithm for the edge cover problem: independently and in parallel, each node $v \in V$ chooses one neighbour $x(v) \in V$ with $\{v, x(v)\} \in E$. The set $C = \{\{v, x(v)\} : v \in V\}$ is a factor 2 approximation of a minimum edge cover. This generalises to the minimum-weight edge cover as well: each node chooses the cheapest edge.

The edge cover problem is a special case of the set cover problem with $\Delta_V = 2$. The trivial 2-approximation algorithm for the edge cover problem can be applied to approximating set cover as well: each customer $k \in K$ chooses independently and in parallel which agent $v \in V$ covers it. The approximation factor is Δ_V.

2.7.8 Local verification and locally checkable proofs

Korman et al. [96, 97] study the problem of *verifying* a solution with a local algorithm. We have seen that the problem of finding a spanning tree is inherently non-local, and if we are given a spanning tree in a natural way as a subset of edges, a local algorithm cannot verify whether it really is a spanning tree or not; recall Figure 2.9 on page 24.

However, it is possible to give a spanning tree together with a *proof* that can be verified with a local algorithm, so that an invalid input is detected by at least one node (assuming that the communication graph $\mathcal{G}$ is connected). For example, we can orient the spanning tree towards an arbitrary root node. For each node, the proof consists of (i) the identity of the root node, (ii) the distance to the root node in the spanning tree, and (iii) the edge that points towards the root node.

2.7.9 Other problems

Kuhn et al. [107] studies a generalisation of covering LPs, with upper bounds for variables x_v.

Our work in Papers I and II presents local algorithms for max-min linear programs; see Chapter 3 for details.

Andersen et al. [5] study local algorithms for *PageRank* [24] computations in the web graph.

A weak 2-colouring also determines a cut with at least $|E|/\Delta$ edges. This gives a partial answer to Elkin's [48] question regarding the distributed approximability of the maximum cut problem.

Positive results for the circuit complexity class NC^0 [7, 37, 81] may also have positive implications in the field of local algorithms; this possible connection calls for further research.

2.8 Randomised local algorithms

There are two key techniques which make randomness a practical tool in the design of centralised algorithms, both familiar from textbooks on randomised algorithms [132, 136]. First, the probability of a "failure" – the event of producing an infeasible output – can be made negligible by adapting the algorithm to the global properties of the input; for example, the number of iterations can depend on the size of the input. Second, it may be possible to detect incorrect output and re-execute the algorithm until the output is correct, turning a Monte Carlo algorithm into a Las Vegas one.

In a local setting, neither of these two techniques can be applied as such. First, the execution of a local algorithm cannot depend on the size of the input. Second, it is not possible to gather the output in a central location for inspection and re-execute the algorithm depending on whether the output is incorrect. Indeed, if a randomised local algorithm has a nonzero probability of failure given some input, then we can simply take several copies of the input to boost the probability that the algorithm makes a failure in at least one copy; see, e.g., Kuhn [102, §4.5].

There are strong negative results on the power of randomness in the local setting. Linial [121] shows that randomness does not help in local algorithms where the objective is to colour a graph, and Kuhn [102, §4.5] extends this to the problem of colour reduction. Naor and Stockmeyer [138] show that randomness does not help in a local algorithm for any locally checkable labelling; this includes, among others, the problem of finding a maximal matching or a maximal independent set in a bounded-degree graph.

Nevertheless, there are positive results where a randomised local algorithm provides a probabilistic approximation guarantee. For example, in some cases it is possible to give an upper bound on the expected approximation factor – here the expectation is over the random coin tosses made by the algorithm; nothing is assumed about the input. If we are content with a probabilistic approximation guarantee, it is possible to overcome some of the negative results in Section 2.6.

Interestingly, Kuhn [102, §2.7.1] shows that probabilistic approximation guarantees do not help with linear programs: if there is a local, randomised algorithm with the expected approximation factor α, then there is a local, deterministic α-approximation algorithm as well. Therefore all positive examples in this section involve combinatorial problems.

2.8.1 Matching and independent set

As we have seen in Section 2.6.3, packing problems such as maximum matching and maximum independent set do not admit deterministic, local approximation algorithms, not even in the case of an n-cycle with unique node identifiers. With these problems, randomness clearly helps.

Wattenhofer and Wattenhofer [164] present a randomised local approximation algorithm for the maximum-weight matching problem in trees; the expected weight of the matching is within factor 4 of the optimum. Hoepman et al. [77] improve the expected approximation ratio to $2 + \varepsilon$.

Nguyen and Onak [139] present a randomised local algorithm for the maximum matching problem in bounded-degree graphs; the approximation ratio is $1 + \varepsilon$ with high probability.

Czygrinow et al. [39] present a randomised local algorithm for finding a maximum independent set in a planar graph; the approximation ratio is $1 + \varepsilon$ with high probability.

2.8.2 Maximum cut and maximum satisfiability

Another negative result from Section 2.6.3 shows that there is no deterministic local approximation algorithm for the maximum cut problem. Again, a randomised local algorithm exists. In this case, we can resort to a very simple algorithm, familiar from introductory courses to randomised algorithms: flip a fair coin for each node to determine its side [126, 132, 136]. The expected size of the cut is $|E|/2$; hence the expected approximation ratio is 2. The algorithm is clearly local; no communication is needed.

A similar approach can be applied to the *maximum satisfiability* (MAX-SAT) problem: choose a random assignment [132, 136]. See, e.g., Ausiello et al. [13, Problem LO1] or Garey and Johnson [60, Problem LO5] for the definition of the problem. However, unlike the maximum cut problem, MAX-SAT has a simple, local, deterministic 2-approximation algorithm: First, for each clause, remove all but one of the literals; in essence, we arrive at a MAX-1-SAT instance. Second, satisfy at least half of the clauses by using a local version of Johnson's [87] 2-approximation algorithm.

2.8.3 LP rounding

Kuhn et al. [102, 107, 108, 110] present a general framework for designing randomised local algorithms for the set cover and set packing problems in bounded-degree graphs. The solution is obtained in three phases: (1) Solve the LP relaxation of the problem approximately with a local algorithm.

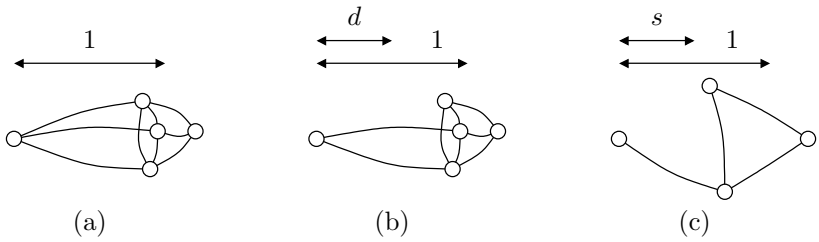

Figure 2.13: (a) A unit-disk graph. (b) A quasi unit-disk graph, with $d = 1/2$. (c) A civilised graph, with $s = 1/2$. This is also a quasi unit-disk graph, with $d < s$.

(2) Apply randomised rounding to find a candidate solution; at this point, the solution is integral but it is not necessarily feasible. (3) Apply a deterministic algorithm to make the solution feasible.

As discussed earlier in Section 2.7, the LP relaxations of the set cover and set packing problems have local approximation schemes in bounded-degree graphs. Together with these algorithms, the LP rounding scheme yields the expected approximation ratio $O(\log \Delta_V)$ for the set cover problem and $O(\Delta_V)$ for the set packing problem.

In bounded-degree graphs, these results imply the following expected approximation ratios: $O(\log \Delta)$ for vertex covers, $O(\log \Delta)$ for dominating sets, $O(\Delta)$ for maximum independent sets, and $O(1)$ for maximum matchings.

2.9 Geometric problems

In this section, we review local algorithms for geometric graphs. In a geometric graph, each node is embedded in a low-dimensional space, typically in the two-dimensional plane.

2.9.1 Models

Most research has focused on the case where $\mathcal{G}$ is a *unit-disk graph*: a pair of nodes is connected by an edge if and only if the Euclidean distance between them is at most 1. See Figure 2.13a.

Work has also been done on generalisations of unit-disk graphs. A *quasi unit-disk graph* [18, 112] is a graph where the nodes are embedded in the two-dimensional plane, the length of an edge is at most 1, and nodes which are within distance d from each other are always connected by an edge; here $0 < d < 1$ is a constant. See Figure 2.13b. A *civilised graph* (graph drawn

in a civilised manner) [47, §8.5] is a graph where the nodes are embedded in the two-dimensional plane, the length of an edge is at most 1, and the distance between any pair of nodes is at least s; here $0 < s < 1$ is a constant. See Figure 2.13c.

By definition, a civilised graph with parameter s is a quasi unit-disk graph with parameter $d < s$; therefore all positive results for quasi unit-disk graphs apply directly to civilised graphs. Furthermore, a civilised graph is a bounded-degree graph, as can be seen by a simple packing argument.

2.9.2 Partial geometric information

For some problems, it is sufficient to have a local knowledge of the embedding. Kuhn et al. [102, 106] show that packing and covering LPs admit local constant-factor approximation algorithms in unit-disk graphs. It is enough that the distances between the nodes are known so that each node can construct a *local coordinate system*.

Our work in Papers III and IV studies scheduling problems in a semi-geometric setting in which the coordinates of the nodes are not known, but a small amount of symmetry-breaking information is available. See Chapter 4 for details.

Most positive results, however, assume that there is a global coordinate system and each node knows its coordinate (so-called *location-aware* nodes). We review these results in the following.

2.9.3 Algorithms from simple tilings

A simple approach for designing local algorithms in a geometric setting is to partition the two-dimensional plane into rectangles, and colour the rectangles with a constant number of colours [72, 73, 102, 106, 134, 168, 170, 171]. Partitioning the two-dimensional plane into rectangles also partitions the network into clusters. If each node knows its coordinates, it knows into which cluster it belongs to.

As a concrete example, we can partition the plane into rectangles of size 2×1 and colour them with 3 colours so that the distance between a pair of nodes in two different rectangles of the same colour is larger than 1. See Figure 2.14 for illustration; we use the names *white*, *light*, and *dark* for the colours.

By construction, we know that if the nodes u and v are in two different rectangles of the same colour, then there is no edge $\{u, v\}$ in a (quasi) unit-disk graph. Furthermore, a packing argument shows that there is a constant upper bound D on the diameter of a connected component of a cluster.

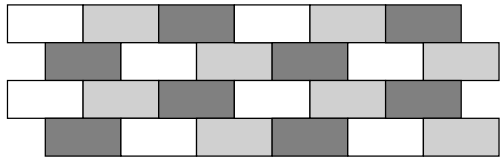

Figure 2.14: Partitioning the two-dimensional plane into 3-coloured rectangles with dimensions 2×1.

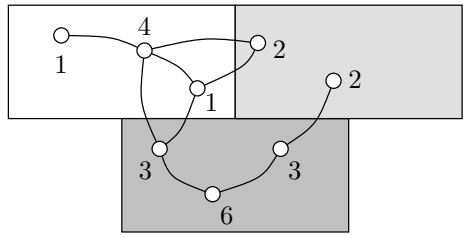

Figure 2.15: Factor 3 approximation for vertex colouring. The edges that cross the boundaries of the tiles can be safely ignored in the algorithm.

In Awerbuch et al.'s [14] terminology, these coloured rectangles provide a $(3, D)$-decomposition of the network. In Attiya et al.'s [11] terminology, the coloured rectangles provide a t-orientation of the graph $\mathcal{G}$ for $t = 3D$.

Now it is easy to design a local 3-approximation algorithm for vertex colouring [72, 102, 168, 170]. We handle each connected component in each cluster independently in parallel. A local algorithm finds an optimal vertex colouring within each component; the components have bounded diameter and hence a local algorithm can gather full information about the component. Connected components in white rectangles assign colours $1, 4, 7, \ldots$, connected components in light rectangles assign colours $2, 5, 8, \ldots$, and connected components in dark rectangles assign colours $3, 6, 9, \ldots$. Put together, we obtain a feasible vertex colouring, and the number of colours that we use is within factor 3 of the optimum; see Figure 2.15.

A similar idea (with larger 3-coloured rectangles) can be used to design local 3-approximation algorithms for the following problems: edge colouring [72], vertex cover [72, 73, 168], and dominating set [72, 73, 168]. These are examples of local algorithms that unscrupulously exploit the assumption that local computation is free; nevertheless, Hunt et al. [80] show how to solve the subproblem of finding a minimum-size dominating set or vertex cover within a rectangle in polynomial time.

Another algorithm design technique that employs the same 3-coloured tiling is the sequential greedy strategy. Consider, for example, the task of finding a maximal independent set. We can proceed in three phases as follows [11, 14, 72]. First, each white rectangle finds a maximal independent set with a greedy algorithm. Then each light rectangle extends the independent set greedily, taking into account the output in neighbouring white rectangles. Finally, each dark rectangle extends the independent set greedily, taking into account neighbouring white and light rectangles. The same technique can be applied to find a maximal matching [72, 171] and a vertex $(\Delta + 1)$-colouring [11, 14, 72]. A local algorithm for maximal matching then gives a 2-approximation of a minimum vertex cover as well.

Finally, it is possible to find a factor 4 approximation of a maximum independent set by using a similar 3-coloured tiling [72, 168]. First, each cluster finds a maximum-size independent set in parallel. This may cause conflicts. The conflicts are then resolved; first those that involve white rectangles and then those that involve light rectangles. At each conflict resolution we lose at most one half of the nodes; hence the remaining nodes provide a factor 4 approximation.

2.9.4 Other algorithms

Wiese and Kranakis [168, 171–173] present local approximation schemes for dominating sets, connected dominating sets, vertex covers, maximum matchings, and maximum independent sets in unit-disk graphs.

Czyzowicz et al. [40] present a 5-approximation algorithm for the dominating set problem and a 7.46-approximation algorithm for the connected dominating set problem. Wiese and Kranakis [168, 169] study local approximation algorithms with local horizon $r \leq 2$ for dominating sets, connected dominating sets, vertex covers, and independent sets in unit-disk graphs. Kuhn and Moscibroda [104] present a local approximation algorithm for the capacitated dominating set problem in unit-disk graphs; this is a variant of the dominating set problem in which each node has a limited capacity that determines how many neighbours it can dominate.

Šparl and Žerovnik [157] present a 4/3-approximation algorithm for multicolouring hexagonal graphs.

2.9.5 Planar subgraphs and geographic routing

In geographic routing [61, 179], it is of interest to construct a connected planar subgraph $\mathcal{H} = (V, E')$ of a unit-disk graph $\mathcal{G} = (V, E)$, with the original set of nodes V but a smaller set of edges $E' \subset E$.

There are local algorithms for constructing planar subgraphs. For example, Gabriel graphs [58] and relative neighbourhood graphs [86, 160] can be constructed with simple local rules.

Once we have constructed a planar subgraph of a unit-disk graph, it is possible to route messages with local geometric rules, assuming that we know the coordinates of the target node [22, 100].

2.9.6 Spanners

In applications such as topology control, merely having a connected planar subgraph $\mathcal{H}$ is not enough. Among others, it is desirable that $\mathcal{H}$ is a *geometric t-spanner*. In a t-spanner, for any pair u, v of nodes in $\mathcal{G}$, the shortest path between u and v in $\mathcal{H}$ is at most t times as long as the shortest path between u and v in $\mathcal{G}$; here the length of a path is the sum of the Euclidean lengths of the edges. The constant t is the *stretch factor* of the spanner.

Gabriel graphs and relative neighbourhood graphs are not t-spanners for any constant t. Yao graphs [176] and θ-graphs [92] provide classical examples of spanners that can be constructed with a simple local algorithm. However, these constructions lack some desirable properties; in particular, they do not have a constant upper bound on the node degree.

Examples of more recent work include the following. Wang and Li [163] present a local algorithm for constructing a planar, bounded-degree spanner in unit-disk graphs. The local algorithms by Li et al. [120] and Kanj et al. [90] further guarantee that the total edge length of the spanner is at most a constant factor larger than the total edge length of a minimum spanning tree. Chávez et al. [31] generalise the results by Li et al. [120] to quasi unit-disk graphs. Wattenhofer and Zollinger [165] present a local algorithm that can be applied in arbitrary weighted graphs, not only in geometric graphs.

2.9.7 Coloured subgraphs

Local algorithms have also been presented for constructing coloured subgraphs. Urrutia [161] presents a local algorithm that constructs a connected, planar, edge-coloured subgraph of a unit-disk graph. Wiese and Kranakis [168, 170] present a local algorithm that constructs a connected, planar, vertex-coloured subgraph of a unit-disk graph. Czyzowicz et al. [41] present a local algorithm for colouring the nodes in an arbitrary planar subgraph of a unit-disk graph. Czyzowicz et al. [42] present a local algorithm for colouring the edges in certain subgraphs of unit-disk graphs.

2.10 Open problems

We conclude this chapter with some open problems related to deterministic local algorithms. We recall that in this work a local algorithm refers to a constant-time algorithm.

Problem 2.1. Is there a local approximation scheme for general packing LPs or covering LPs in bounded-degree graphs?

The local approximation scheme by Kuhn et al. [108] assumes not only a degree bound but also an upper bound for the ratio of largest coefficient to smallest coefficient in the LP. Techniques by Luby and Nisan [127] and Bartal et al. [19] can be applied to avoid the dependency on coefficients, but this comes at the cost of adding a dependency on the size of the input [109].

Problem 2.2. Is it possible to achieve a better approximation ratio than Δ_V and $\Delta_K + \varepsilon$ for the set cover problem with a deterministic local algorithm?

In a centralised setting, the greedy algorithm [30, 87, 123] achieves the approximation ratio $1 + 1/2 + \cdots + 1/\Delta_V = O(\log \Delta_V)$. The randomised local algorithm [102, 108] achieves the expected approximation factor $O(\log \Delta_V)$. However, no deterministic local algorithm is known with the approximation factor better than Δ_V.

Problem 2.3. Is there a combinatorial packing problem that admits a nontrivial, deterministic, local approximation algorithm?

Finding a simple 2-matching is a packing problem, but it is a slightly contrived example. It would be interesting to see other, more natural examples of packing problems that can be solved locally, without any auxiliary information.

A partial answer is provided by the local approximation algorithm for the maximum matching problem, based on the weak 2-colouring algorithm by Naor and Stockmeyer [138]. However, this can be applied only in a graph where every node has an odd degree, a rather stringent assumption.

Problem 2.4. Is there a problem that (i) can be solved with a local algorithm that exploits the numerical values of the identifiers, and (ii) cannot be solved with an order-invariant local algorithm that merely compares the identifiers?

Naor and Stockmeyer [138] show that order-invariant local algorithms are sufficient for locally checkable labellings: if there is a local algorithm

for a locally checkable labelling problem, then there is an order-invariant algorithm as well. Hence we need to seek for an example outside locally checkable labellings.

If we assume that the set of node identifiers is $\{1, 2, \ldots, |V|\}$, then we can find some examples of problems that admit a local algorithm and do not admit an order-invariant local algorithm. For example, in this case leader election is trivial with a local algorithm (the node number 1 is the leader) but there is no order-invariant local algorithm for the task. However, this example is no longer valid if the unique node identifiers are an arbitrary subset of, say, $\{1, 2, \ldots, 2|V|\}$.

Chapter 3

Local algorithms for max-min LPs

This chapter is based on Papers I and II.

3.1 Introduction

Recall the definition of a max-min LP in Section 2.4.4: the objective is to maximise ω subject to $A\mathbf{x} \leq \mathbf{1}$, $C\mathbf{x} \geq \omega\mathbf{1}$, and $\mathbf{x} \geq \mathbf{0}$ for non-negative matrices A and C. The name "max-min LP" comes from the following equivalent formulation:

$$\begin{aligned} &\text{maximise} && \min_{k \in K} \mathbf{c}_k \mathbf{x} && \\ &\text{subject to} && \mathbf{a}_i \mathbf{x} \leq 1 && \text{for each } i \in I, \\ & && x_v \geq 0 && \text{for each } v \in V. \end{aligned}$$

Each row $\mathbf{a}_i$ of A has at most Δ_I positive elements, and each row $\mathbf{c}_k$ of C has at most Δ_K positive elements.

This chapter summarises the results of Papers I and II; the work is related to local algorithms and max-min linear programs in distributed systems.

3.2 Applications

We begin with motivating examples that illustrate how max-min LPs arise in distributed systems. Our main example stems from the task of data gathering in wireless sensor networks.

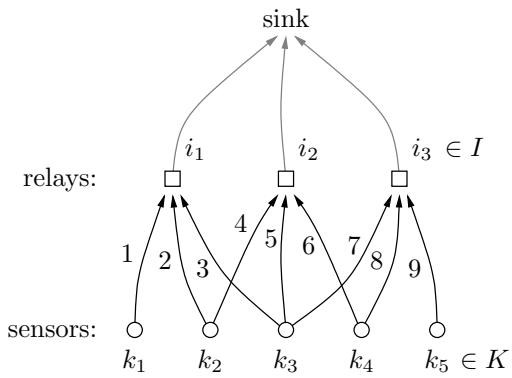

Figure 3.1: A sensor network (based on Figure 1.1 on page 2).

3.2.1 Data gathering in sensor networks

Consider the wireless sensor network illustrated in Figure 3.1. In this simple example, there are 5 sensors, $k_1, k_2, \ldots, k_5$. Each sensor produces data, which needs to be gathered in the sink node. The sensor nodes have a low-power radio, with which they cannot reach the sink node directly. Hence we use a two-tier topology: there are 3 relay nodes, i_1, i_2, i_3, which can be used to forward data from the sensor nodes to the sink.

The edges denote possible data flows in the network. For example, data generated by the sensor k_1 can only be forwarded through the relay i_1. However, for the sensor k_2, there are several possibilities: we can transfer a part of the data through the relay i_1 and the rest of the data through the relay i_2.

We use the non-negative variables $x_1, x_2, \ldots, x_9$ to denote the amount of data transmitted on the edges $1, 2, \ldots, 9$, respectively. For example, the variable x_6 is the amount of data transmitted from the sensor k_4 through the relay i_2 to the sink. The total amount of data forwarded by the relay i_2 is hence $x_4 + x_5 + x_6$, and the total amount of data gathered from the sensor k_2 is $x_2 + x_4$.

To keep this example simple, we assume that the bottleneck is the limited battery capacity of the relay nodes – long-distance data transmissions between the relay and the sink consume a large amount of energy. We choose the units so that each relay can forward 1 unit of data in total. That is, we need to choose data flows $\mathbf{x} \geq \mathbf{0}$ that satisfy the packing constraints $x_1 + x_2 + x_3 \leq 1$, $x_4 + x_5 + x_6 \leq 1$, and $x_7 + x_8 + x_9 \leq 1$.

To arrive at a well-defined optimisation problem, we need to define the objective function that characterises the utility or goodness of a feasible

solution $\mathbf{x}$. Naturally the choice of the objective function depends on the application, but there are two simple candidates that have been studied in the literature. The first candidate is to maximise the *total amount* of data gathered [78, 153], that is,

$$\begin{aligned} \text{maximise} \quad & x_1 + x_2 + \cdots + x_9 \\ \text{subject to} \quad & x_1 + x_2 + x_3 \leq 1, \\ & x_4 + x_5 + x_6 \leq 1, \\ & x_7 + x_8 + x_9 \leq 1, \\ & x_1, x_2, \ldots, x_9 \geq 0. \end{aligned} \tag{3.1}$$

The second candidate is to maximise the *minimum amount* of data gathered per sensors [54, 89], that is

$$\begin{aligned} \text{maximise} \quad & \min\{x_1,\ x_2 + x_4,\ x_3 + x_5 + x_7,\ x_6 + x_8,\ x_9\} \\ \text{subject to} \quad & x_1 + x_2 + x_3 \leq 1, \\ & x_4 + x_5 + x_6 \leq 1, \\ & x_7 + x_8 + x_9 \leq 1, \\ & x_1, x_2, \ldots, x_9 \geq 0. \end{aligned} \tag{3.2}$$

Note that the formulation in (3.2) is equivalent to maximising the *lifetime* of the network, assuming that each sensor node produces data at the same constant rate and all data must be routed to the sink.

The optimisation problem (3.1) is a packing LP, and local algorithms from prior work [102, 108, 143] can be used to solve it approximately in a distributed setting. However, this formulation alone does not capture well the requirements of a typical sensor network [54]. For example, an optimal solution of (3.1) gathers 1 unit of data from each of the sensors k_1, k_2, k_3 and completely ignores the sensors k_4, k_5.

The optimisation problem (3.2) is arguably a more faithful representation of the real-world goals of a typical sensor network: no sensor is ignored. In any optimal solution of (3.2), we receive $3/5$ units of data from each sensor $k_1, k_2, \ldots, k_5$. An example of an optimal solution is

$$\begin{aligned} x_1 = x_5 = x_9 &= 3/5, \\ x_2 = x_8 &= 2/5, \\ x_4 = x_6 &= 1/5, \\ x_3 = x_7 &= 0. \end{aligned}$$

However, the problem (3.2) is a max-min LP, and prior work on packing or covering LPs cannot be applied to solve it. In this chapter, we study local algorithms for solving this kind of optimisation problems.

3.2.2 Fair bandwidth allocation

Other optimisation problems in communication networks have a structure similar to Figure 3.1, and they also lead to max-min LPs similar to (3.2). For example, consider the task of allocating a fair share of bandwidth to each customer of an Internet service provider. Each customer is connected to some wireless access points in the vicinity, and each access point has a limited-bandwidth connection to the Internet. The objective is to maximise the minimum bandwidth allocated to each customer, subject to the bandwidth constraints. This setting is essentially equal to the example in Section 3.2.1, if we replace the sensors with customers and the relays with access points.

3.2.3 Mixed packing and covering

On a more abstract level, a local approximation algorithm for max-min LPs is also a local algorithm for *approximate mixed packing and covering*. A mixed packing and covering problem is a system of inequalities of the form

$$\begin{aligned} A\mathbf{x} &\leq \mathbf{1}, \\ C\mathbf{x} &\geq \mathbf{1}, \\ \mathbf{x} &\geq \mathbf{0} \end{aligned} \tag{3.3}$$

for non-negative matrices A and C. In α-approximate mixed packing and covering [178] the task is to either (i) find an *approximately feasible* solution $\mathbf{x} \geq \mathbf{0}$ such that $A\mathbf{x} \leq \alpha\mathbf{1}$ and $C\mathbf{x} \geq \mathbf{1}$, or (ii) prove that the original problem (3.3) does not have a solution.

Now assume that we have a local α-approximation algorithm $\mathcal{A}$ for max-min LPs. Using $\mathcal{A}$, we can find an α-approximation to the problem of maximising ω subject to $A\mathbf{x} \leq \alpha\mathbf{1}$, $C\mathbf{x} \geq \omega\alpha\mathbf{1}$, and $\mathbf{x} \geq \mathbf{0}$. We make two observations.

1. If there is a vector $\mathbf{x}$ that satisfies (3.3), the output of the algorithm $\mathcal{A}$ provides an approximately feasible solution $\mathbf{x} \geq \mathbf{0}$ such that $A\mathbf{x} \leq \alpha\mathbf{1}$ and $C\mathbf{x} \geq \mathbf{1}$.

2. If the output of $\mathcal{A}$ violates $\mathbf{c}_k\mathbf{x} \geq 1$ for some $k \in K$, then we know that (3.3) does not have a feasible solution. Furthermore, the node k can detect the violation.

Hence we can solve an α-approximate mixed packing and covering problem in the following sense: either (i) we find an approximately feasible solution $\mathbf{x}$, or (ii) at least one node detects that (3.3) does not have a solution.

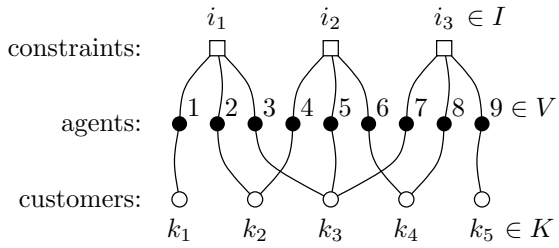

Figure 3.2: The graph $\mathcal{G} = (V \cup I \cup K, E)$ for the max-min LP (3.2).

An application of mixed packing and covering mentioned by Young [178] is related to non-negative systems of linear equations. Given a non-negative matrix A, the task is to find a non-negative $\mathbf{x}$ such that $A\mathbf{x} = \mathbf{1}$. If we can solve the α-approximate mixed packing and covering problem, we can also find a non-negative $\mathbf{x}$ such that $\mathbf{1} \leq A\mathbf{x} \leq \alpha\mathbf{1}$, or show that there is no non-negative solution to $A\mathbf{x} = \mathbf{1}$.

3.3 Definitions

As defined in Section 2.4.4, we assume the following distributed setting. The communication graph is a bipartite graph $\mathcal{G} = (V \cup I \cup K, E)$ where edges $e \in E$ are of the form $e = \{v, i\}$ or $e = \{v, k\}$ with $v \in V$ (agents), $i \in I$ (constraints), and $k \in K$ (customers). A positive coefficient $a_{iv} > 0$ implies that $\{i, v\} \in E$, and a positive coefficient $c_{kv} > 0$ implies that $\{k, v\} \in E$.

3.3.1 An example of the communication graph

For the problem (3.2), the graph $\mathcal{G}$ is illustrated in Figure 3.2. Comparing Figure 3.1 with Figure 3.2, it is evident that the graph $\mathcal{G}$ is closely related to the physical structure of the original sensor network. In particular, a short path in $\mathcal{G}$ corresponds to a small number of sensor–relay links in the original network. A local algorithm in the graph $\mathcal{G}$ can be efficiently implemented in the sensor network of Figure 3.1, and we do not need to use any of the long-distance links between the relays and the sink.

In this example, the customers $k \in K$ and the constraints $i \in I$ correspond to physical computational entities in the network (sensor nodes and relay nodes, respectively). This is not the case for the agents $v \in V$; however, the behaviour of an agent can be *simulated* by a physical entity, for example, by the adjacent relay node.

In the following, we focus on the abstract communication graph $\mathcal{G}$. The algorithms are written as if each node of $\mathcal{G}$ was a computational entity.

3.3.2 Bipartite and 0/1 max-min LPs

The max-min LP (3.2) has two special properties, both of which turn out to simplify the design of local algorithms.

First, each element of A and C is either 0 or 1. Using the definition from Section 2.4.4, we say that (3.2) is a 0/1 *max-min LP*.

Second, each column of A has exactly 1 positive element, and each column of C has exactly 1 positive element as well. Put otherwise, in the graph $\mathcal{G}$, each agent $v \in V$ is adjacent to exactly one constraint $i \in I$ and exactly one customer $k \in K$. Such a special case is called a *bipartite max-min LP*.

The name "bipartite max-min LP" refers to the fact that our starting point was a bipartite graph, with edges between the customers and the constraints (black arrows in Figure 3.1). We then replaced each edge with a customer–agent–constraint path. By definition, this procedure leads to a bipartite max-min LP, with each agent adjacent to one customer and one constraint. It should be noted that the graph $\mathcal{G}$ is always bipartite, both in the case of bipartite max-min LPs and non-bipartite max-min LPs.

A simple two-tier structure – such as the sensors and the relays in the example of Section 3.2.1, or the customers and the access points in Section 3.2.2 – leads to a bipartite max-min LP. A more complicated network, with bottlenecks on several levels of the hierarchy, would give rise to a more general non-bipartite max-min LP.

3.4 Background

Papadimitriou and Yannakakis [143] present a simple local approximation algorithm for packing LPs, the so-called *safe algorithm*; it turns out that this is a local approximation algorithm for max-min LPs as well.

3.4.1 The safe algorithm

In the safe algorithm, the agent v chooses

$$x_v = \min_{i \in I:\, \{i,v\} \in E} \frac{1}{a_{iv} \deg(i)}.$$

Intuitively, each constraint $i \in I$ divides its "capacity" evenly among all its neighbours: if i has $\deg(i)$ neighbours, each neighbour $v \in V$ of i can use

at most $1/\deg(i)$ of the capacity – that is, $a_{iv}x_v$ is at most $1/\deg(i)$. In particular, the node v chooses the largest possible value x_v that does not violate these allotments for any of its adjacent constraints.

Clearly this is a feasible solution to a packing LP or a max-min LP: by construction, none of the packing constraints are violated. Now let $\mathbf{x}^*$ be an optimal solution. Then it holds that $a_{iv}x_v^* \leq 1$ for all $i \in I$ and $v \in V$, that is,

$$x_v^* \leq \min_{i \in I\colon \{i,v\} \in E} \frac{1}{a_{iv}}.$$

Therefore $x_v \geq x_v^*/\Delta_I$ for all $v \in V$. In particular, x is a Δ_I-approximation to a packing LP. Furthermore, it is a Δ_I-approximation to a max-min LP, and the solution does not depend on the matrix C at all. The local horizon of this local algorithm is 1.

3.4.2 Simple special cases

The special cases of $\Delta_I = 1$ or $\Delta_K = 1$ can be solved optimally with a simple local algorithm. To see this, note that the safe algorithm produces an optimal solution in the case $\Delta_I = 1$. Let us then present a local algorithm for the case $\Delta_K = 1$. For each agent $v \in V$, constraint $i \in I$, and customer $k \in K$, we use the shorthand notation

$$\begin{aligned} I(v) &= \{i \in I : \{i, v\} \in E\}, \\ K(v) &= \{k \in K : \{k, v\} \in E\}, \\ V(i) &= \{v \in V : \{i, v\} \in E\}, \\ V(k) &= \{v \in V : \{k, v\} \in E\}. \end{aligned}$$

By assumption, $|V(k)| \leq 1$ for each $k \in K$. If $V(k) = \emptyset$, the problem is trivial: the row $\mathbf{c}_k$ of the matrix C is equal to zero, the optimum of the max-min LP is $\omega = 0$, and any feasible solution is optimal. Hence we can focus on the case $|V(k)| = 1$ for all $k \in K$. Furthermore, if $K(v) = \emptyset$ for an agent $v \in V$, we can set $x_v = 0$; therefore we focus on the case $K(v) \neq \emptyset$ for all $v \in V$.

Now the sets $K(v)$ form a partition of K, and we can write the utility $\omega(\mathbf{x})$ of a solution $\mathbf{x}$ as

$$\omega(\mathbf{x}) = \min_{k \in K} \mathbf{c}_k \mathbf{x} = \min_{v \in V} \min_{k \in K(v)} c_{kv} x_v = \min_{v \in V} x_v \min_{k \in K(v)} c_{kv} = \min_{v \in V} c(v) x_v,$$

where $c(v) = \min_{k \in K(v)} c_{kv}$. Therefore there is an optimal solution $\mathbf{x}^*$ with utility ω such that $c(v)x_v^* = \omega$ for all $v \in V$. Let $v \in V$ be an agent and let

$i \in I(v)$ be an adjacent constraint; the optimal solution $\mathbf{x}^*$ satisfies

$$c(v)x_v^* \sum_{u \in V(i)} \frac{a_{iu}}{c(u)} = \omega \sum_{u \in V(i)} \frac{a_{iu}}{c(u)} = \sum_{u \in V(i)} a_{iu} x_u^* \leq 1. \tag{3.4}$$

Consider the local algorithm that chooses

$$x_v = c(v)^{-1} \min_{i \in I(v)} \left(\sum_{u \in V(i)} \frac{a_{iu}}{c(u)} \right)^{-1} \tag{3.5}$$

for each agent $v \in V$. First, this choice is feasible, as

$$\sum_{v \in V(i)} a_{iv} x_v \leq \sum_{v \in V(i)} \frac{a_{iv}}{c(v)} \left(\sum_{u \in V(i)} \frac{a_{iu}}{c(u)} \right)^{-1} = 1$$

for each constraint $i \in I$. Second, this choice is optimal, as $x_v \geq x_v^*$ for each agent $v \in V$ by (3.4) and (3.5).

Hence the case $\Delta_I = 1$ or $\Delta_K = 1$ can be solved optimally with a simple local algorithm. For the rest of this chapter, we focus on the non-trivial case $\Delta_I \geq 2$ and $\Delta_K \geq 2$.

3.5 Our results

As we saw in Section 3.4.1, the same local algorithm, the safe algorithm, can be used to obtain a Δ_I-approximation for both packing LPs and max-min LPs. However, from the perspective of local approximability, the commonalities between packing LPs and max-min LPs end here. For packing and covering LPs in bounded-degree graphs, the approximation factor can be improved to $1 + \varepsilon$ for any $\varepsilon > 0$, at the cost of increasing the local horizon [102, 108]. For max-min LPs, this is not the case; our work in Papers I and II presents lower bounds for the approximation factor achievable with any local algorithm.

3.5.1 Lower bounds

The first lower bound is from Paper I. While this is not yet a tight bound, it already establishes that there is no local approximation scheme for max-min LPs in general.

Theorem 3.1. *For any $\Delta_I \geq 2$ and $\Delta_K \geq 2$, there is no local approximation algorithm for max-min LPs with an approximation ratio less than $\Delta_I/2 + 1/2 - 1/(2\Delta_K - 2)$.*

Subsequently, the result was improved in Paper II:

Theorem 3.2. *For any $\Delta_I \geq 2$ and $\Delta_K \geq 2$, there is no local approximation algorithm for max-min LPs with the approximation ratio equal to $\Delta_I(1 - 1/\Delta_K)$.*

This lower bound holds even in the case of a bipartite 0/1 max-min LP, and with globally unique node identifiers given as input.

Theorem 3.2 leaves a small gap between the approximation factor Δ_I that can be achieved with the safe algorithm, and the approximation factor $\Delta_I(1 - 1/\Delta_K)$ that cannot be achieved with any local algorithm. It turns out that Theorem 3.2 is tight; to close the gap between the positive and negative results, we need improved local approximation algorithms.

3.5.2 Upper bounds

In Paper II, we present a local algorithm that closes the gap between the positive and negative results for all bipartite max-min LPs:

Theorem 3.3. *For any $\Delta_I \geq 2$, $\Delta_K \geq 2$, and $\varepsilon > 0$, there is a local approximation algorithm for bipartite max-min LPs with the approximation ratio $\Delta_I(1 - 1/\Delta_K) + \varepsilon$.*

The algorithm assumes only a port numbering; hence unique node identifiers are not needed to achieve the best possible approximation factor for bipartite max-min LPs. The algorithm from Theorem 3.3 is based on the following ingredients.

1. Graph *unfolding* (see Section 2.5.1) is used to simplify the structure of the problem. We show that if there is a local α-approximation for bipartite max-min LPs in (infinite) trees in the port numbering model, then there is a local α-approximation for bipartite max-min LPs in general graphs as well. Hence it is enough to focus on the special case of trees.

2. The tree is *regularised*, so that each constraint has degree exactly Δ_I and each customer has degree exactly Δ_K.

3. A *local subproblem* is constructed for each customer $k \in K$, based on a constant-radius neighbourhood of k in the regularised tree. This is a max-min LP of a constant size, and it can be solved optimally with an LP solver.

4. Each agent is involved in several local subproblems. The agent computes the *average* of these solutions.

5. Finally, the value is *scaled down* by a constant q to obtain a feasible solution.

Our latest work [53] shows that Theorem 3.2 is tight for non-bipartite max-min LPs as well:

Theorem 3.4. *For any $\Delta_I \geq 2$, $\Delta_K \geq 2$, and $\varepsilon > 0$, there is a local approximation algorithm for max-min LPs with the approximation ratio $\Delta_I(1 - 1/\Delta_K) + \varepsilon$.*

3.5.3 Bounded relative growth

From a practical perspective, the negative result of Theorem 3.2 is discouraging, in particular in the case of a large Δ_I. However, the constructions that we use to derive the negative results are based on regular trees – or, more accurately, regular high-girth graphs [79, 117, 125, 131]. The *girth* is the length of a shortest cycle; regular high-girth graphs look *locally* like regular trees, as there is no short cycle.

In a regular tree, the size of $B_{\mathcal{G}}(v, r)$, the radius-r neighbourhood of the node v, is exponential in r. However, in a typical large-scale deployment of a wireless sensor network, it is likely that the growth of $B_{\mathcal{G}}(v, r)$ is only polynomial in r; after all, if there is an upper bound on the density of the sensors in the 2 or 3-dimensional space, and an upper bound on the range of the radio, it is not possible to fit more than $\mathrm{poly}(r)$ sensors within r hops from a device. As a simple example, consider a regular d-dimensional grid; in such a graph, the size of $B_{\mathcal{G}}(v, r)$ is proportional to r^d.

In this section, we investigate the local approximability of max-min LPs in graphs where the growth of $B_{\mathcal{G}}(v, r)$ resembles grid graphs, at least for sufficiently large values of r. We say that the graph $\mathcal{G}$ has *bounded relative growth $1 + \delta$ beyond radius R* if

$$\frac{|V \cap B_{\mathcal{G}}(v, r + 2)|}{|V \cap B_{\mathcal{G}}(v, r)|} \leq 1 + \delta$$

for all $v \in V$ and $r \geq R$. Note that here we focus on the number of agents; hence the increment in the formula is 2, as the distance between a pair of agents is always even. If $\mathcal{G}$ is a regular d-dimensional grid, then for any $\delta > 0$ there is an R such that $\mathcal{G}$ has bounded relative growth $1 + \delta$ beyond radius R.

In Paper I, we present an algorithm that finds a good approximation for max-min LPs in graphs with small relative growth:

Theorem 3.5. *For any R, there is a local approximation algorithm for max-min LPs with the approximation ratio $(1+\delta)^2$ in graphs that have bounded relative growth $1+\delta$ beyond radius R.*

The local horizon of the algorithm is linear in R. This is a local approximation scheme for max-min LPs in grid graphs and other families of graphs in which δ approaches 0, as R increases.

For a small δ, the leading terms in the approximation factor of Theorem 3.5 are $1+2\delta$. In Paper II, we investigate whether this is close to the best possible. We prove the following lower bound.

Theorem 3.6. *Let $\Delta_I \geq 3$, $\Delta_K \geq 3$, and $0 < \delta < 1/10$, and assume that the graph $\mathcal{G}$ has bounded relative growth $1+\delta$ beyond some constant radius R. Then there is no local approximation algorithm for max-min LPs with an approximation ratio less than $1+\delta/2$.*

Hence the coefficient of δ in the approximation factor cannot be improved by a factor larger than 4; in particular, there is no local $(1+o(\delta))$-approximation algorithm for max-min LPs in graphs with bounded relative growth $1+\delta$. Informally, the algorithm of Theorem 3.5 makes good use of the structure of graphs with bounded relative growth.

Chapter 4

Local algorithms for scheduling

This chapter is based on Papers III and IV.

4.1 Introduction

In this chapter, we study local algorithms for *scheduling problems*. In a distributed scheduling problem, the objective is to choose a *schedule* for each node. A schedule is a list of time intervals; the time intervals indicate when the node is active. We focus on two different kinds of scheduling problems: sleep scheduling, which is a maximisation problem, and activity scheduling, which is a minimisation problem.

We begin with an informal introduction to these two problems in Sections 4.2 and 4.3. We will give the formal definitions in Section 4.4.

4.2 Sleep scheduling

The motivation for studying the sleep scheduling problem comes from the task of maximising the lifetime of a wireless sensor network. Consider the network illustrated in Figure 4.1a; the graph $\mathcal{G} = (V, E)$ shows the communication links in the network. Each sensor node $v \in V$ is powered by a battery with capacity $b(v)$. In this example, $b(v) = 1$ for each node $v \in V$; this means that the node v can operate for 1 time unit (for example, 1 year).

4.2.1 Redundancy graphs

If all sensor nodes are switched on at the same time, then all batteries drain in 1 time unit. The lifetime of the network is 1. However, it may be

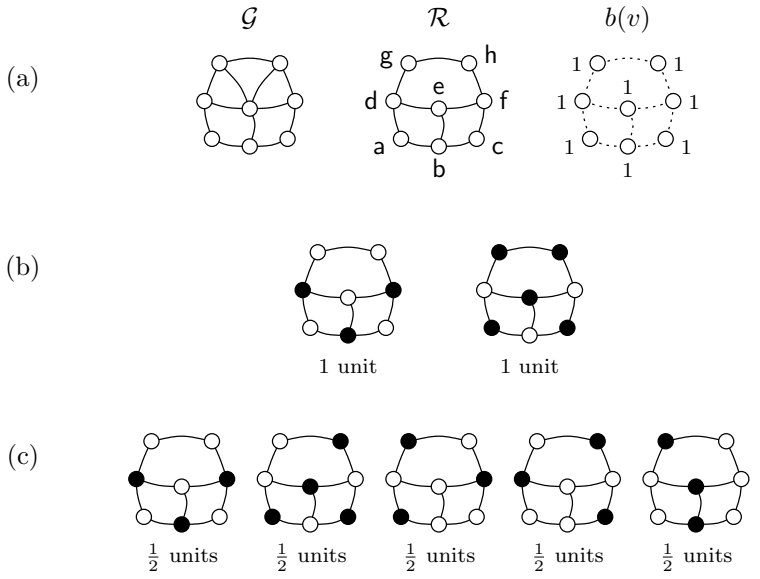

Figure 4.1: (a) An instance of the sleep scheduling problem. (b) An optimal integral solution (a maximum domatic partition); a black node is awake and a white node is asleep. (c) A fractional solution (feasible but not optimal).

possible to achieve a longer lifetime by exploiting *redundancy* in the sensor network: we can put redundant nodes into sleep mode.

In this chapter, we focus on *pairwise redundancy* [25, 99, 135, 146]. We say that sensors u and v are pairwise redundant if the node u can be asleep whenever v is awake and vice versa. For example, the nodes u and v may be physically so close to each other that the measurements from u and v are highly correlated, and in our application, we only need measurements from one of them.

Pairwise redundancy can be represented as a *redundancy graph* $\mathcal{R}$; see Figure 4.1a for an example. An edge $\{u, v\}$ in the graph $\mathcal{R}$ denotes that u and v are pairwise redundant. We assume that the redundancy graph $\mathcal{R}$ is a subgraph of the communication graph $\mathcal{G}$: if there is no radio link between a pair of nodes, it is not likely that they are pairwise redundant either.

4.2.2 Examples of schedules

Figure 4.1b shows a way to achieve a longer lifetime by exploiting pairwise redundancy. The illustration shows a *sleep schedule* of length 2. First, for 1 time unit, a set of 3 nodes is awake – all other nodes can be asleep as

they are redundant. Second, for 1 time unit, a set of 5 nodes is awake and all other nodes are asleep. Each node is awake for exactly 1 time unit in total; the lifetime of the network is 2 units.

If $D \subseteq V$ is a valid set of nodes that is awake simultaneously, then D is a dominating set in the redundancy graph $\mathcal{R}$, and vice versa. Indeed, in a dominating set D any node $v \notin D$ (asleep) is adjacent to at least one node $u \in D$ (awake). For example, Figure 4.1b shows two dominating sets of the graph $\mathcal{R}$. Furthermore, these dominating sets are disjoint; they form a domatic partition of $\mathcal{R}$ (see Section 2.4.1 for definitions).

If each battery has capacity 1, and if we cannot temporarily switch off a node once we have switched it on, then an optimal domatic partition provides the longest possible schedule – note that in this case it is never advantageous to switch on a node at a non-integral point in time, say, at time $t = 0.5$, as we can equally well wait until time $t = 1$. The domatic partition in Figure 4.1b is optimal; to establish the optimality, note that there is no dominating set of size 2 in $\mathcal{R}$. Hence each dominating set has size at least 3; as there are only 8 nodes in the graph, it is not possible to find more than 2 disjoint dominating sets.

However, if we can switch a node on and off several times during its lifetime, it may be possible to find a better solution (cf. Berman et al. [20]); this chapter focuses on such schedules. Figure 4.1c shows an example. Here we have 5 dominating sets, and each of them is active for 1/2 time units. In total, each individual node is awake for 1 time unit, and the lifetime of the network is $5/2 = 2.5$ units. For example, the node d is switched on at time 0, switched off at time 1/2, switched on again at time 3/2, and finally drains its battery at time 2.

But is it possible to achieve an even longer lifetime? To address this question, let us formulate the problem as a linear program.

4.2.3 LP formulation

We need to choose a (possibly empty) time period for each dominating set. Without loss of generality, we can focus on *minimal* dominating sets – a dominating set is minimal if none of its proper subsets is a dominating set. In the redundancy graph $\mathcal{R}$ of Figure 4.1a, there are 17 minimal dominating sets: $\{\mathsf{a},\mathsf{b},\mathsf{h}\}$, $\{\mathsf{a},\mathsf{c},\mathsf{e},\mathsf{g}\}$, $\{\mathsf{a},\mathsf{c},\mathsf{e},\mathsf{h}\}$, $\{\mathsf{a},\mathsf{d},\mathsf{f}\}$, $\{\mathsf{a},\mathsf{f},\mathsf{g}\}$, $\{\mathsf{a},\mathsf{f},\mathsf{h}\}$, $\{\mathsf{b},\mathsf{c},\mathsf{g}\}$, $\{\mathsf{b},\mathsf{d},\mathsf{f}\}$, $\{\mathsf{b},\mathsf{d},\mathsf{h}\}$, $\{\mathsf{b},\mathsf{e},\mathsf{g}\}$, $\{\mathsf{b},\mathsf{e},\mathsf{h}\}$, $\{\mathsf{b},\mathsf{f},\mathsf{g}\}$, $\{\mathsf{b},\mathsf{g},\mathsf{h}\}$, $\{\mathsf{c},\mathsf{d},\mathsf{f}\}$, $\{\mathsf{c},\mathsf{d},\mathsf{g}\}$, $\{\mathsf{c},\mathsf{d},\mathsf{h}\}$, and $\{\mathsf{d},\mathsf{e},\mathsf{f}\}$.

The order of the time periods is arbitrary; we are interested in choosing the lengths of the time periods. Let, for example, x_{abh} denote the length of the time period associated with the dominating set $\{\mathsf{a},\mathsf{b},\mathsf{h}\}$. With this

notation, the task is to

$$\begin{aligned}
\text{maximise} \quad & x_{\mathsf{abh}} + x_{\mathsf{aceg}} + x_{\mathsf{aceh}} + \cdots + x_{\mathsf{def}} \\
\text{subject to} \quad & x_{\mathsf{abh}} + x_{\mathsf{aceg}} + x_{\mathsf{aceh}} + x_{\mathsf{adf}} + x_{\mathsf{afg}} + x_{\mathsf{afh}} \leq 1, \\
& x_{\mathsf{abh}} + x_{\mathsf{bcg}} + x_{\mathsf{bdf}} + x_{\mathsf{bdh}} + x_{\mathsf{beg}} + x_{\mathsf{beh}} + x_{\mathsf{bfg}} + x_{\mathsf{bgh}} \leq 1, \\
& x_{\mathsf{aceg}} + x_{\mathsf{aceh}} + x_{\mathsf{bcg}} + x_{\mathsf{cdf}} + x_{\mathsf{cdg}} + x_{\mathsf{cdh}} \leq 1, \\
& x_{\mathsf{adf}} + x_{\mathsf{bdf}} + x_{\mathsf{bdh}} + x_{\mathsf{cdf}} + x_{\mathsf{cdg}} + x_{\mathsf{cdh}} + x_{\mathsf{def}} \leq 1, \\
& x_{\mathsf{aceg}} + x_{\mathsf{aceh}} + x_{\mathsf{beg}} + x_{\mathsf{beh}} + x_{\mathsf{def}} \leq 1, \\
& x_{\mathsf{adf}} + x_{\mathsf{afg}} + x_{\mathsf{afh}} + x_{\mathsf{bdf}} + x_{\mathsf{bfg}} + x_{\mathsf{cdf}} + x_{\mathsf{def}} \leq 1, \\
& x_{\mathsf{aceg}} + x_{\mathsf{afg}} + x_{\mathsf{bcg}} + x_{\mathsf{beg}} + x_{\mathsf{bfg}} + x_{\mathsf{bgh}} + x_{\mathsf{cdg}} \leq 1, \\
& x_{\mathsf{abh}} + x_{\mathsf{aceh}} + x_{\mathsf{afh}} + x_{\mathsf{bdh}} + x_{\mathsf{beh}} + x_{\mathsf{bgh}} + x_{\mathsf{cdh}} \leq 1, \\
& x_{\mathsf{abh}}, x_{\mathsf{aceg}}, x_{\mathsf{adf}}, \ldots, x_{\mathsf{def}} \geq 0.
\end{aligned}$$

For example, the first constraint ensures that the node a is awake for at most 1 time unit in total.

Solving the LP, we find out that the optimum is $13/5 = 2.6$. To see that the optimum is at least $13/5$, consider the feasible solution $x_{\mathsf{def}} = 3/5$, $x_{\mathsf{abh}} = x_{\mathsf{afh}} = x_{\mathsf{bcg}} = x_{\mathsf{cdg}} = 2/5$, $x_{\mathsf{aceh}} = x_{\mathsf{beg}} = 1/5$, and all other variables equal to 0. To see that the optimum is at most $13/5$, consider the dual and its feasible solution $y_{\mathsf{a}} = y_{\mathsf{c}} = y_{\mathsf{e}} = 1/5$ and $y_{\mathsf{b}} = y_{\mathsf{d}} = y_{\mathsf{f}} = y_{\mathsf{g}} = y_{\mathsf{h}} = 2/5$.

4.3 Activity scheduling

In this section, we introduce the activity scheduling problem. The motivation for studying the problem comes from the task of completing a set of activities in parallel in the shortest possible time – for example, transmitting a set of messages in a wireless network in the shortest possible time.

Figure 4.2a illustrates such a scenario. In the communication graph $\mathcal{G} = (V, E)$, each node $v \in V$ needs to be active for $a(v)$ units of time. In this example, $a(v) = 1$ for each v.

4.3.1 Conflict graphs

If the activities of the nodes do not interfere with each other, then we can simply let all nodes be active simultaneously and complete all activities in a total of 1 time unit. However, the task becomes non-trivial if the activities of the nodes conflict with each other – for example, the activities may be radio transmissions close to each other on the same frequency band.

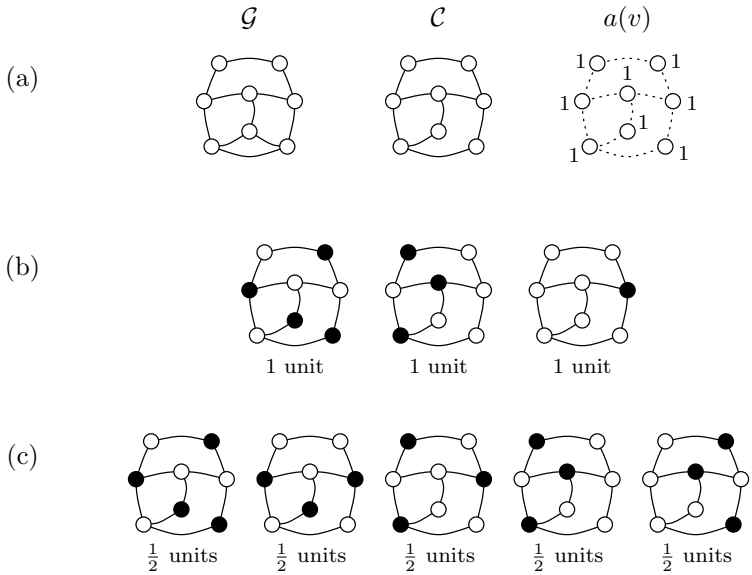

Figure 4.2: (a) An instance of the activity scheduling problem. (b) An optimal integral solution; a black node is active and a white node is inactive. (c) An optimal fractional solution.

We will focus on *pairwise conflicts* [84]: we say that the activities of the nodes u and v conflict with each other if u must be inactive whenever v is active and vice versa. Again, such pairwise relations can be represented as a graph, a *conflict graph* $\mathcal{C}$; again, we will assume that the conflict graph $\mathcal{C}$ is a subgraph of the communication graph $\mathcal{G}$. See Figure 4.2a for an illustration.

4.3.2 Examples of schedules

Figure 4.2b shows how to complete the activities in 3 time units: first a set of 4 nodes is active for 1 time units, then a set of 3 nodes is active for 1 time unit and finally a set of 1 node is active for 1 time unit. Each node is active for 1 time unit in total; that is, each node is able to complete its activities.

If $I \subseteq V$ is a set of nodes that are active simultaneously, then I is an independent set in the conflict graph $\mathcal{C}$, and vice versa. All three sets illustrated in Figure 4.2b are independent sets of $\mathcal{C}$; furthermore, they form a partition of the vertex set V. Put otherwise, Figure 4.2b illustrates a vertex 3-colouring of the conflict graph $\mathcal{C}$; each independent set is one colour

class. Furthermore, as there is an odd cycle in $\mathcal{C}$, we know that there is no 2-colouring of $\mathcal{C}$. That is, we cannot partition V into 2 independent sets.

However, if we can switch nodes on and off more than once, then it is possible to find a schedule that is strictly shorter than 3 time units; see Figure 4.2c for an example that shows how to complete all activities in 5/2 time units. The solution is optimal: there is a 5-cycle in $\mathcal{C}$, and any independent set contains at most 2 of these 5 nodes; hence at least 5/2 units of time is required until all nodes of the 5-cycle have completed their activities.

4.4 Definitions

Now we proceed to give the formal definitions of the scheduling problems that we study in this chapter. Both problems can be formulated as linear programs.

4.4.1 Sleep scheduling

Let $\mathcal{G} = (V, E)$ be a communication graph and let $\mathcal{R}$ be a redundancy graph, a subgraph of $\mathcal{G}$. Associated with each $v \in V$ is a capacity $b(v) \geq 0$. In the sleep scheduling problem, the task is to

$$\begin{aligned} \text{maximise} \quad & \sum_{D} x(D) \\ \text{subject to} \quad & \sum_{D\colon v \in D} x(D) \leq b(v) \qquad \text{for each } v \in V, \\ & x(D) \geq 0 \qquad \text{for each } D. \end{aligned} \tag{4.1}$$

Here D ranges over all dominating sets of $\mathcal{R}$, and $x(D)$ is the length of the time period associated with the dominating set D. The LP (4.1) is a generalisation of the example in Section 4.2.3.

In a local setting, we cannot find an explicit solution $\mathbf{x}$ to (4.1); we use the following implicit representation of the solution. In a local algorithm for sleep scheduling, the local output of a node $v \in V$ is a schedule for the node v – a set of time intervals during which the node v is awake. The total length of these time intervals must be at most $b(v)$. We say that a local algorithm produces a sleep schedule of length T if the following holds for all $0 \leq t \leq T$ and for all nodes $v \in V$: if the node v is asleep at time t, then there is at least one node u adjacent to v in $\mathcal{R}$ such that u is awake at time t. Such a schedule exists if and only if the optimum of (4.1) is at least T.

Note that we explicitly allow that some nodes are awake even after time T; from the perspective of applications, these is merely a good thing. In general, a local algorithm cannot know the lifetime T; a bottleneck for the lifetime can be in a distant part of the network.

4.4.2 Activity scheduling

Let $\mathcal{G} = (V, E)$ be a communication graph and let $\mathcal{C}$ be a conflict graph, a subgraph of $\mathcal{G}$. Associated with each $v \in V$ is a requirement $a(v) \geq 0$. In the activity scheduling problem, the task is to

$$\begin{aligned} \text{minimise} \quad & \sum_I x(I) \\ \text{subject to} \quad & \sum_{I\colon v \in I} x(I) \geq a(v) \qquad \text{for each } v \in V, \\ & x(I) \geq 0 \qquad \text{for each } I. \end{aligned} \tag{4.2}$$

Here I ranges over all independent sets of $\mathcal{C}$, and $x(I)$ is the length of the time period associated with the independent set I.

Again, in a local setting, the output of a node $v \in V$ is a schedule for the node v – a set of time intervals during which the node v is active. The total length of these time intervals must be at least $a(v)$. We say that a local algorithm produces an activity schedule of length T if the following holds for all nodes $v \in V$: the node v is not active after the time T; furthermore, if the node v is active at a time $0 \leq t \leq T$, then each node u adjacent to v in $\mathcal{C}$ is inactive at time t. Again, such a schedule exists if and only if the optimum of (4.2) is at most T.

4.5 Background

The special cases of $b(v) = 1$ and $a(v) = 1$ are closely related to classical optimisation problems. First, consider the integer program

$$\begin{aligned} \text{maximise} \quad & \sum_D x(D) \\ \text{subject to} \quad & \sum_{D\colon v \in D} x(D) \leq 1 \qquad \text{for each } v \in V, \\ & x(D) \in \{0, 1\} \qquad \text{for each } D, \end{aligned} \tag{4.3}$$

where D ranges over all dominating sets of $\mathcal{R}$. In essence, (4.3) is a reformulation of the maximum domatic partition problem: the dominating

sets D with $x(D) = 1$ are, by construction, disjoint. The optimum of (4.3) is the *domatic number* of $\mathcal{R}$ [32, 51]. Hence (4.1) is, in the case $b(v) = 1$, an LP relaxation of the domatic partition problem.

Second, consider the integer program

$$\begin{aligned} \text{minimise} \quad & \sum_I x(I) \\ \text{subject to} \quad & \sum_{I\colon v \in I} x(I) \geq 1 \qquad \text{for each } v \in V, \\ & x(I) \in \{0,1\} \qquad \text{for each } I, \end{aligned} \tag{4.4}$$

where I ranges over all independent sets of $\mathcal{C}$. This integer program is a re-formulation of the minimum vertex colouring problem: the independent sets I with $x(I) = 1$ cover all vertices, and each such set is one colour class. The optimum of (4.4) is the *chromatic number* of $\mathcal{C}$. Hence (4.2) is, in the case $a(v) = 1$, an LP relaxation of the vertex colouring problem.

4.5.1 Terminology

The LP relaxation of (4.4) is called the *fractional vertex colouring* (or fractional graph colouring) problem, and the optimum is the *fractional chromatic number* of the graph. Following the analogy between domination and colouring [32], we use the term *fractional domatic partition* for the LP relaxation of (4.3), and the term *fractional domatic number* for its optimum. Hence the sleep scheduling problem (4.1) is a generalisation of the fractional domatic partition problem, and the activity scheduling problem (4.2) is a generalisation of fractional vertex colouring.

4.5.2 Centralised algorithms

The LP (4.1) is a packing LP, and the LP (4.2) is a covering LP (see Sections 2.4.2 and 2.4.3). However, even in a centralised setting these problems are not easy to solve. In the worst case, the number of variables in (4.1) or (4.2) is exponential in $|V|$: there is one variable for each (minimal) dominating set of $\mathcal{R}$ in (4.1) and one variable for each (maximal) independent set of $\mathcal{C}$ in (4.2).

It turns out that both of these scheduling problems are computationally hard to solve and even hard to approximate in general graphs. For example, Lund and Yannakakis [128] show that both vertex colouring and fractional vertex colouring are NP-hard to approximate within factor $|V|^{\varepsilon}$ for some positive ε.

Some special cases admit centralised polynomial-time algorithms. For example, if the conflict graph $\mathcal{C}$ is the line graph of a given graph $\mathcal{H}$, then an independent set in $\mathcal{C}$ corresponds to a matching in $\mathcal{H}$ and vice versa. In this case the activity scheduling problem can be solved in polynomial time [69].

4.6 Marked graphs

As we have seen in Section 2.6.3, a local algorithm cannot find a non-trivial approximation to dominating set or independent set; this is true even in the case of cycle graphs. As a direct consequence, there is no local algorithm for sleep scheduling or activity scheduling with a good approximation factor.

Some auxiliary information is needed to break the symmetry. Knowing the coordinates of the nodes is one possibility (see Section 2.9); however, having a globally consistent coordinate system is a rather stringent assumption. In this chapter, we show that a relatively weak assumption is enough: some nodes are designated as *marker nodes*. Paper III gives semi-geometric requirements for the choice of the markers; in this chapter, we focus on the strictly combinatorial definition in Paper IV.

We say that the graph $\mathcal{G} = (V, E)$ together with a set of markers $M \subseteq V$ is a $(\Delta, \ell_1, \ell_\mu, \mu)$-*marked graph* if for each $v \in V$, (i) the degree of v in $\mathcal{G}$ is at most Δ, (ii) $M \cap B_{\mathcal{G}}(v, \ell_1) \neq \emptyset$, and (iii) $|M \cap B_{\mathcal{G}}(v, \ell_\mu)| \leq \mu$.

Put otherwise, a marked graph is a bounded-degree graph where there is always at least one marker within distance ℓ_1 from any node, and at most μ markers within distance ℓ_μ from any node; here $\ell_\mu > \ell_1$. Ideally, μ is small and $\ell_\mu - \ell_1$ is large. Figure 4.3a shows a marked graph with 4 markers.

4.7 Our results

Paper IV presents local approximation algorithms for the scheduling problems in marked graphs. Here the communication graph $\mathcal{G}$ is required to be a marked graph; the redundancy graph $\mathcal{R}$ and the conflict graph $\mathcal{C}$ can be arbitrary subgraphs of $\mathcal{G}$. The main results are summarised by the following theorems.

Theorem 4.1. *There is a local* $(1 + \varepsilon)$*-approximation algorithm for sleep scheduling in* $(\Delta, \ell_1, \ell_\mu, \mu)$*-marked graphs for any* $\varepsilon > 4\Delta/\lfloor(\ell_\mu - \ell_1)/\mu\rfloor$.

Theorem 4.2. *There is a local* $1/(1 - \varepsilon)$*-approximation algorithm for activity scheduling in* $(\Delta, \ell_1, \ell_\mu, \mu)$*-marked graphs for any* $\varepsilon > 4/\lfloor(\ell_\mu - \ell_1)/\mu\rfloor$.

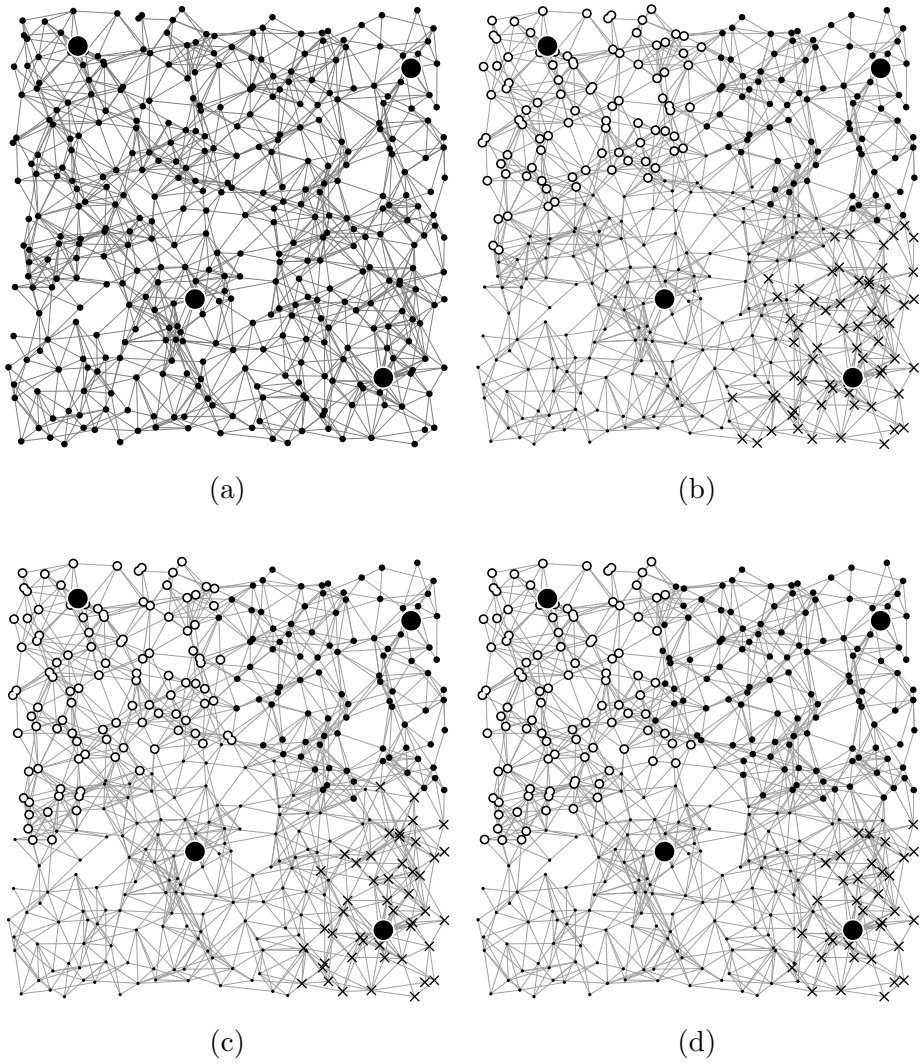

Figure 4.3: (a) A communication graph with four markers. (b) Configuration 0; four cells are denoted by four different symbols. The cells are Voronoi cells for the markers in the metric of the communication graph. (c)–(d) Configurations 1 and 2. In comparison with the configuration 0, each cell boundary is shifted in a particular direction, depending on the identifiers of the markers.

Thanks to Topi Musto for help with preparing the illustration.

Paper IV builds on the techniques introduced in Paper III. The technique can be interpreted as an adaptation of the classical *shifting strategy* [17, 75] into a local, coordinate-free setting.

As illustrated in Figure 4.3b, the Voronoi cells for the markers can be used to partition the nodes of the communication graph into cells; the size of a cell is bounded by a constant (that depends on ℓ_1 and Δ). For each cell, we can define a subproblem, solve it optimally with a local algorithm, and then put together the schedules obtained from the subproblems. This indeed produces a solution where the nodes in the interior of each cell are performing at least as well as in the optimum. However, the nodes near the cell boundaries perform suboptimally, as there is no global coordination between the cells.

To alleviate the problem, we can shift the node boundaries as illustrated in Figures 4.3b–4.3d. For each pair of adjacent cells, the boundary is moved in a direction that is determined based on the unique identifiers of the marker nodes; for the details of the shifting technique, see Paper IV. We construct a number of such configurations. The key observation is the following: for each node $v \in V$, the number of configurations where v is near *any* cell boundary is small in comparison with the total number of configurations (see Lemmas 2–4 in Paper IV).

We solve the local subproblems for each configuration; thus we obtain one globally consistent schedule for each configuration. In the worst case, none of these schedules is near-optimal in itself: in each of them, the nodes near the cell boundaries may perform suboptimally. However, if we *interleave* all these schedules, we even out the suboptimalities, as no single node is too often near a cell boundary. The same basic approach can be applied to both the sleep scheduling problem and the activity scheduling problem. In essence, we exploit the linearity of (4.1) and (4.2).

The results of this chapter highlight the trade-off between computational efficiency and ease of deployment. If we do not install marker nodes or any other symmetry-breaking devices, we have a multitude of negative results for constant-time distributed algorithms (see Section 2.6). However, if we invest some extra effort in the network deployment and place some marker nodes in appropriate locations, then it is possible to design local algorithms for problems such as sleep scheduling and activity scheduling.

Chapter 5

Local graphs

This chapter is based on Papers V and VI.

5.1 Introduction

In this chapter, we study *centralised* algorithms instead of distributed algorithms. Whenever we mention an approximation algorithm, we refer to a *polynomial-time* approximation algorithm. A *polynomial-time approximation scheme* or a PTAS is a family of polynomial-time algorithms such that for each $\varepsilon > 0$ there is a $(1 + \varepsilon)$-approximation algorithm.

We begin with the definitions of local graphs (see Paper VI) and local conflict graphs (see Paper V). The focus is on graphs embedded in the two-dimensional space $\mathbb{R}^2$. In the following, $\|x - y\|$ is the Euclidean distance between $x \in \mathbb{R}^2$ and $y \in \mathbb{R}^2$. A *unit disk* is a disk of radius 1, i.e., a unit disk centred at x consists of the points y with $\|x - y\| \leq 1$.

5.1.1 Local graphs

For a positive integer N, an *N-local graph* is a graph $\mathcal{G} = (V, E)$ in which each node $v \in V$ is associated with a unique point $p(v) \in \mathbb{R}^2$ so that

(i) any unit disk contains at most N points in $p(V)$,

(ii) each edge $\{u, v\} \in E$ satisfies $\|p(u) - p(v)\| \leq 1$.

The definition of local graphs is similar in nature to civilised graphs (see Section 2.9.1): edges are not too long, and nodes are not deployed in a too dense manner. However, in a local graph, there may be a pair of nodes very close to each other (assuming $N \geq 2$). Hence a local graph is not necessarily a civilised graph, unless we change the embedding of the nodes.

The definition of an N-local graph is motivated by the redundancy graphs that we studied in Section 4.2.1. Each node $v \in V$ is a sensor that is placed at the location $p(v)$. In this context, both assumptions are arguably realistic:

(i) sensor nodes are not deployed in an arbitrarily dense manner,

(ii) if the nodes u and v are pairwise redundant, then u and v are close to each other.

We do not assume that u and v are pairwise redundant whenever they are close to each other. We merely assume that nodes placed far from each other cannot be pairwise redundant.

5.1.2 Local conflict graphs

As a generalisation of local graphs, we define the family of local conflict graphs. An *N-local conflict graph* is a graph $\mathcal{C} = (V, E)$ in which each node $v \in V$ is associated with a unique pair of points $\tau(v), \rho(v) \in \mathbb{R}^2$ so that

(i) any unit disk contains at most N points in $\tau(V) \cup \rho(V)$,

(ii) each edge $\{u, v\} \in E$ satisfies $\|\tau(u) - \rho(v)\| \leq 1$ or $\|\tau(v) - \rho(u)\| \leq 1$.

In the special case $\tau(v) = \rho(v)$ for all v, an N-local conflict graph is an N-local graph. However, the family of N-local conflict graphs is much larger than the family of N-local graphs. For example, we show in Paper V that any bipartite graph can be represented as a 1-local conflict graph. Furthermore, an arbitrary graph on at most N^2 vertices can be represented as an N-local conflict graph.

The definition of an N-local conflict graph is motivated by the conflict graphs that we studied in Section 4.3.1. Each node $v \in V$ corresponds to a radio transmission; the transmitter is placed at the location $\tau(v)$ and the receiver is placed at the location $\rho(v)$. The focus is on radio networks where the radio interference is dominated by the near-far effect: a radio receiver may be blocked by another transmitter close to it. In this context, both assumptions are arguably realistic:

(i) radio transceivers are not deployed in an arbitrarily dense manner,

(ii) if the radio transmissions u and v interfere with each other, then the transmitter of u is close to the receiver of v or vice versa.

Again, we do not assume that interference occurs whenever two devices are close to each other. We merely assume that there is no interference between a distant pair of a transmitter and a receiver.

5.2 Maximum-weight independent set

In this section, we revisit the activity scheduling problem that we defined in Section 4.4.2. However, this time we approach the problem from the perspective of centralised algorithms; the aim is to design an algorithm that can be applied in any N-local conflict graph $\mathcal{C}$, for any constant N.

If we could afford exponential time, we could simply enumerate all (maximal) independent sets of the conflict graph $\mathcal{C}$ and solve the LP (4.2). However, we aim at a polynomial-time approximation algorithm.

5.2.1 Activity scheduling and oracles

It turns out that the problem of approximating the LP (4.2) is closely related to the problem of finding a *maximum-weight independent set* in the conflict graph $\mathcal{C} = (V, E)$. In the maximum-weight independent set problem, each node $v \in V$ is associated with a positive weight $w(v)$. The objective is to find an independent set $I \subseteq V$ in $\mathcal{C}$ that maximises the total weight $w(I) = \sum_{v \in I} w(v)$.

If we had an oracle that returns a maximum-weight independent set of $\mathcal{C}$ for any given weight vector $\mathbf{w}$, then we could use, e.g., the approximation scheme by Young [178] to find an approximately optimal solution to the activity scheduling problem (4.2); the number of oracle invocations is polynomial in the size of the input.

Finding a maximum-weight or maximum-size independent set is an NP-hard problem. However, it turns out that it is sufficient to have an oracle that finds an α-approximation of a maximum-weight independent set; see, e.g., the general results by Jansen [85]. If the oracle runs in a polynomial time, this yields a polynomial-time $(\alpha+\varepsilon)$-approximation algorithm for the activity scheduling problem for any $\varepsilon > 0$.

5.2.2 Our results

The independent set problem is prohibitively hard to approximate in general graphs [82, 94]; hence some structural assumptions on the conflict graph $\mathcal{C}$ are needed in order to find a polynomial-time constant-factor approximation algorithm.

In Paper V, we show that it is enough to assume that $\mathcal{C}$ is an N-local conflict graph, for some constant N.

Theorem 5.1. *For any constants N and $\varepsilon > 0$, there is a polynomial-time $(5 + \varepsilon)$-approximation algorithm for the maximum-weight independent set problem in N-local conflict graphs.*

The approximation factor does not depend on the constant N; the value of N affects only the running time of the algorithm.

The algorithmic techniques in the proof of Theorem 5.1 include an application of the shifting strategy [17, 75, 80] to construct subproblems. The subproblems are solved in polynomial time by a combination of two approaches: exhaustive search and a greedy algorithm.

As a complement to Theorem 5.1, we also show that the problem of finding a maximum-weight independent set in a local conflict graph does not admit a PTAS, unless P = NP. The proof is by a reduction from the directed cut problem [142].

5.3 Identifying and locating–dominating codes

In this section, we turn our attention to identifying codes and locating–dominating codes. We begin by introducing a shorthand notation. Let $\mathcal{G} = (V, E)$ be a simple undirected graph, and let $C \subseteq V$ be a subset of nodes. For each node $v \in V$, we define

$$C(v) = C \cap B_{\mathcal{G}}(v, 1),$$

the set of elements in C within the radius-1 neighbourhood of v. Using this notation, we recall that C is a dominating set (see Section 2.4.1) if and only if $C(v) \neq \emptyset$ for all $v \in V$.

Now we are ready to define the codes that we study in this section. We say that C is a *locating–dominating code* if $C(v) \neq \emptyset$ for all $v \in V$, and $C(u) \neq C(v)$ for all $u, v \in V \setminus C$. Furthermore, C is an *identifying code* if $C(v) \neq \emptyset$ for all $v \in V$, and $C(u) \neq C(v)$ for all $u, v \in V$.

Put otherwise, an identifying code is a dominating set in which the subset of dominators $C(v) \subseteq C$ uniquely identifies the node $v \in V$. In a locating–dominating code, we only require that $C(v)$ is unique for the nodes that are not in C.

5.3.1 Examples and applications

Figure 5.1a shows a dominating set that is not a locating–dominating code: we have $C(1) = C(6) = \{2\}$, which contradicts with the definition of a locating–dominating code. Hence it is not an identifying code, either. Figure 5.1b shows a locating–dominating code: the sets $C(2) = \{1, 3\}$, $C(5) = \{4\}$, and $C(6) = \{3\}$ are non-empty and pairwise distinct. However, this is not an identifying code, as we have $C(3) = C(4) = \{3, 4\}$. Finally,

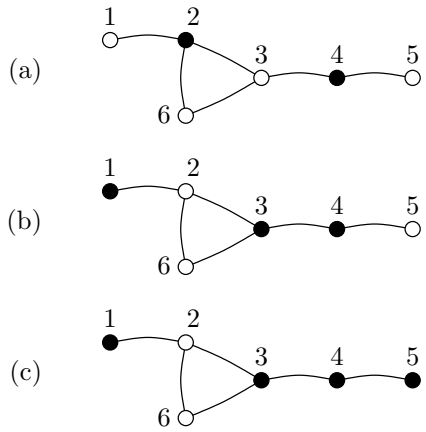

Figure 5.1: The black nodes illustrate (a) a dominating set, (b) a locating–dominating code, and (c) an identifying code.

Figure 5.1c shows an identifying code: the sets $C(1) = \{1\}$, $C(2) = \{1, 3\}$, $C(3) = \{3, 4\}$, $C(4) = \{3, 4, 5\}$, $C(5) = \{4, 5\}$, and $C(6) = \{3\}$ are all non-empty and pairwise distinct.

As an application, assume that the graph $\mathcal{G} = (V, E)$ in Figure 5.1 is a schematic representation of a building. Each node $v \in V$ is a room in which we can place a sensor, for example, a motion detector. Each edge $\{u, v\} \in E$ indicates visibility between the rooms u and v: a motion detector in the room v is triggered if there is motion in the room u and vice versa.

Now assume that we place the motion detectors in a subset $C \subseteq V$ of the rooms. If C is a dominating set, then we know that at least one motion detector is triggered if there is motion in some room $v \in V$. For example, the dominating set in Figure 5.1a shows how to guard the whole building with only two sensors. However, while we can *detect* the intruders, we cannot *locate* them: in the example of Figure 5.1a, if the motion detector is triggered only in the room 2, we do not know whether the intruder is in the room 1, 2, or 6.

An identifying code – for example, the one shown in Figure 5.1c – solves the problem. An identifying code is a dominating set, and hence whenever there is an intruder in the building, we can detect it. But we can also locate the intruder, assuming there is only one intruder. If there is an intruder in the room v, the set $C(v) \subseteq C$ of the motion detectors that are triggered uniquely *identifies* the room v, allowing us to locate the intruder.

If our motion detectors were three-state devices that can distinguish between no motion (no signal), motion in an adjacent room (a weak signal),

and motion in the same room (a strong signal), then the task is slightly easier, and it is sufficient that C is a locating–dominating code.

5.3.2 Background and related work

The definition of a locating–dominating code is due to Slater [156], and identifying codes were introduced by Karpovsky et al. [91]. Related concepts include the *metric dimension* of a graph [60, 95], and the *alarm placement* problem [116, 149].

A locating–dominating code always exists: $C = V$ is an example. It is possible that there is no identifying code; however, if an identifying code exists, then $C = V$ is an identifying code, and hence it is easy to test whether there is an identifying code for a given graph. However, finding a locating–dominating code or identifying code of minimum size is computationally hard.

For a given graph $\mathcal{G}$ and a given integer k, it is NP-hard to decide whether there is an identifying code C with $|C| \leq k$ [28, 33]; the same applies to locating–dominating codes [28, 34] and their directed versions [27]. Polynomial-time algorithms are known for certain special cases [26, 34, 156], and heuristic algorithms [150, 151] have been proposed in the literature. However, in spite of extensive research related to identifying and locating–dominating codes [122], the approximability of identifying codes and locating–dominating codes remained an open question [133, §4.1].

The work in Paper VI resolved the issue. This turned out to be a timely problem to study. The approximability of identifying and locating–dominating codes was investigated independently and in parallel by at least two other groups [66, 67, 114, 115]; see Laifenfeld and Trachtenberg [113, 114] for a summary of the contributions and the complementary approaches taken in the papers.

5.3.3 Our results

It is possible to reduce the problem of finding a minimum-size identifying code or a locating–dominating code to the set cover problem; then we can apply the greedy algorithm for the set cover problem [30, 87, 123]. This results in a polynomial-time approximation algorithm with a logarithmic approximation ratio.

Theorem 5.2. *There are polynomial-time $O(\log |V|)$-approximation algorithms for minimum-size identifying codes and locating–dominating codes.*

However, Paper VI shows that sublogarithmic approximation factors are intractable. It is possible to construct a reduction from the dominating set

problem; this polynomial reduction establishes the following theorem. See, for example, Lund and Yannakakis [128], Raz and Safra [152], and Feige [50] for the inapproximability of the minimum dominating set problem.

Theorem 5.3. *There is a constant $\rho > 0$ such that for any constant $\alpha > 0$, a polynomial-time $(1+\alpha \ln |V|)$-approximation algorithm for minimum-size identifying codes or locating–dominating codes implies a polynomial-time $\max(1, \rho\alpha \ln |V|)$-approximation algorithm for minimum dominating set.*

In bounded-degree graphs, the trivial choice $C = V$ provides a constant-factor approximation. However, there is no PTAS for identifying codes or locating–dominating codes in bounded-degree graphs unless P = NP. This is shown in Paper VI by presenting a reduction from the problem of finding a minimum dominating set in a bounded-degree graph; the problem is known to be APX-hard [93, 142].

Theorem 5.4. *If there is a PTAS for minimum-size identifying codes or locating–dominating codes in bounded-degree graphs, then there is a PTAS for minimum dominating set in bounded-degree graphs.*

However, there is a PTAS in local graphs; the algorithm presented in Paper VI uses the shifting strategy [17, 75, 80].

Theorem 5.5. *For any N and $\varepsilon > 0$, there are polynomial-time $(1+\varepsilon)$-approximation algorithms for minimum-size identifying codes and locating–dominating codes in N-local graphs.*

Paper VI also presents extensions of the results to so-called t-identifying codes and t-locating–dominating codes for a general t, as well as an extension of Theorem 5.5 to more than 2 dimensions (e.g., points are located in $\mathbb{R}^3$ instead of $\mathbb{R}^2$).

Chapter 6

Conclusions

In this thesis we have studied approximation algorithms for optimisation problems that were motivated by large-scale distributed systems such as wireless sensor networks.

The results presented in Section 5 have provided answers to two open questions: the question posed by Jain et al. [84, §A] regarding a structural property of realistic conflict graphs that makes it easy to find maximum independent sets, and the question posed by Moncel [133, §4.1] regarding the approximability of the minimum identifying code problem.

Our results also give rise to a new research question: the approximability of other classical graph problems in local conflict graphs. The shifting strategy [17, 75] can be applied to obtain PTASs for many classical graph problems in families of graphs that are similar to local graphs [80]. However, the same technique cannot be applied to local conflict graphs directly – indeed, we have shown that there is no PTAS for maximum-weight independent sets in local conflict graphs unless P = NP.

The main contributions of this thesis are the local algorithms presented in Chapters 3 and 4. Our work has provided new examples of non-trivial network coordination problems that can be solved with a local approximation algorithm. Put otherwise, we have shown that these global problems can be solved by using only local information: the local outputs of the nodes constitute a solution that is not only feasible but also provably within a constant factor of the global optimum.

In spite of the new positive results, the full capabilities of local algorithms are still far from being fully understood. The question posed by Naor and Stockmeyer [138] in the title of their seminal paper – "What can be computed locally?" – is still largely unanswered. For example, only a few problems in Table 2.4 have matching upper and lower bounds; see also Section 2.10 for other concrete research questions.

A possible path towards a more comprehensive understanding of local algorithms is to borrow powerful techniques from other areas of theoretical computer science. For example, polynomial-time reductions have been widely successful in the study of polynomial-time algorithms and NP-completeness [60], and the idea of reductions can also be applied in the context of local algorithms. Our recent work has provided new examples of the uses of local reductions, both in the design of local algorithms [53], and also as a proof technique for obtaining lower bounds [10]. In the future, we aim to explore this line of research in more depth, and to identify new connections between local algorithms and other areas of theoretical computer science.

References

[1] Scott Aaronson, Greg Kuperberg, and Christopher Granade. Complexity zoo. `http://www.complexityzoo.com/`, July 2008.

[2] Robert P. Adler, Jonathan Huang, Raymond Kong, Philip Muse, Lama Nachman, Rahul C. Shah, Chieh-Yih Wan, and Mark Yarvis. Edge processing and enterprise integration: Closing the gap on deployable industrial sensor networks. In *Proc. 4th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON, San Diego, CA, USA, June 2007)*, pages 620–630. IEEE, Piscataway, NJ, USA, 2007.

[3] Ian F. Akyildiz, Weilian Su, Yogesh Sankarasubramaniam, and Erdal Cayirci. Wireless sensor networks: A survey. *Computer Networks*, 38(4):392–422, 2002.

[4] Alon Amit, Nathan Linial, Jiří Matoušek, and Eyal Rozenman. Random lifts of graphs. In *Proc. 12th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA, Washington, DC, USA, January 2001)*, pages 883–894. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2001.

[5] Reid Andersen, Christian Borgs, Jennifer Chayes, John Hopcraft, Vahab S. Mirrokni, and Shang-Hua Teng. Local computation of PageRank contributions. In *Proc. 5th International Workshop on Algorithms and Models for the Web-Graph (WAW, San Diego, CA, USA, December 2007)*, volume 4863 of *Lecture Notes in Computer Science*, pages 150–165. Springer, Berlin, Germany, 2007.

[6] Dana Angluin. Local and global properties in networks of processors. In *Proc. 12th Annual ACM Symposium on Theory of Computing (STOC, Los Angeles, CA, USA, April 1980)*, pages 82–93. ACM Press, New York, NY, USA, 1980.

[7] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Cryptography in NC^0. *SIAM Journal on Computing*, 36(4):845–888, 2006.

[8] Matti Åstrand, Patrik Floréen, Valentin Polishchuk, Joel Rybicki, Jukka Suomela, and Jara Uitto. Edge dominating sets, simple 2-matchings, and local algorithms, 2009. Unpublished results.

[9] Matti Åstrand, Patrik Floréen, Valentin Polishchuk, Joel Rybicki, Jukka Suomela, and Jara Uitto. A local 2-approximation algorithm for the vertex cover problem, 2009. Manuscript submitted for publication.

[10] Matti Åstrand, Valentin Polishchuk, Joel Rybicki, Jukka Suomela, and Jara Uitto. Local algorithms in (weakly) coloured graphs, 2009. Manuscript.

[11] Hagit Attiya, Hadas Shachnai, and Tami Tamir. Local labeling and resource allocation using preprocessing. *SIAM Journal on Computing*, 28(4):1397–1413, 1999.

[12] Hagit Attiya, Marc Snir, and Manfred K. Warmuth. Computing on an anonymous ring. *Journal of the ACM*, 35(4):845–875, 1988.

[13] Giorgio Ausiello, Pierluigi Crescenzi, Giorgio Gambosi, Viggo Kann, Alberto Marchetti-Spaccamela, and Marco Protasi. *Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties*. Springer, Berlin, Germany, 2003.

[14] Baruch Awerbuch, Andrew V. Goldberg, Michael Luby, and Serge A. Plotkin. Network decomposition and locality in distributed computation. In *Proc. 30th Annual Symposium on Foundations of Computer Science (FOCS, Research Triangle Park, NC, USA, October–November 1989)*, pages 364–369. IEEE, Piscataway, NJ, USA, 1989.

[15] Baruch Awerbuch and Michael Sipser. Dynamic networks are as fast as static networks. In *Proc. 29th Annual Symposium on Foundations of Computer Science (FOCS, White Plains, NY, USA, October 1988)*, pages 206–219. IEEE, Piscataway, NJ, USA, 1988.

[16] Baruch Awerbuch and George Varghese. Distributed program checking: a paradigm for building self-stabilizing distributed protocols. In *Proc. 32nd Annual Symposium on Foundations of Computer Science (FOCS, San Juan, Puerto Rico, October 1991)*, pages 258–267. IEEE, Piscataway, NJ, USA, 1991.

[17] Brenda S. Baker. Approximation algorithms for NP-complete problems on planar graphs. *Journal of the ACM*, 41(1):153–180, 1994.

[18] Lali Barrière, Pierre Fraigniaud, Lata Narayanan, and Jaroslav Opatrny. Robust position-based routing in wireless ad-hoc networks with irregular transmission ranges. *Wireless Communications and Mobile Computing Journal*, 3(2):141–153, 2003.

[19] Yair Bartal, John W. Byers, and Danny Raz. Global optimization using local information with applications to flow control. In *Proc. 38th Annual Symposium on Foundations of Computer Science (FOCS, Miami Beach, FL, USA, October 1997)*, pages 303–312. IEEE Computer Society Press, Los Alamitos, CA, USA, 1997.

[20] Piotr Berman, Gruia Calinescu, Chintan Shah, and Alexander Zelikovsky. Power efficient monitoring management in sensor networks. In *IEEE Wire-*

less Communications and Networking Conference (WCNC, Atlanta, GA, USA, March 2004), pages 2329–2334. IEEE, Piscataway, NJ, USA, 2004.

[21] Paolo Boldi and Sebastiano Vigna. An effective characterization of computability in anonymous networks. In *Proc. 15th International Symposium on Distributed Computing (DISC, Lisbon, Portugal, October 2001)*, volume 2180 of *Lecture Notes in Computer Science*, pages 33–47. Springer, Berlin, Germany, 2001.

[22] Prosenjit Bose, Pat Morin, Ivan Stojmenović, and Jorge Urrutia. Routing with guaranteed delivery in ad hoc wireless networks. *Wireless Networks*, 7(6):609–616, 2001.

[23] Anne Bottreau and Yves Métivier. The Kronecker product and local computations in graphs. In *Proc. 21st International Colloquium on Trees in Algebra and Programming (CAAP, Linköping, Sweden, April 1996)*, volume 1059 of *Lecture Notes in Computer Science*, pages 2–16. Springer, Berlin, Germany, 1996.

[24] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. *Computer Networks and ISDN Systems*, 30(1–7):107–117, 1998.

[25] Mihaela Cardei, David MacCallum, Maggie Xiaoyan Cheng, Manki Min, Xiaohua Jia, Deying Li, and Ding-Zhu Du. Wireless sensor networks with energy efficient organization. *Journal of Interconnection Networks*, 3(3–4):213–229, 2002.

[26] Irène Charon, Sylvain Gravier, Olivier Hudry, Antoine Lobstein, Michel Mollard, and Julien Moncel. A linear algorithm for minimum 1-identifying codes in oriented trees. *Discrete Applied Mathematics*, 154(8):1246–1253, 2006.

[27] Irène Charon, Olivier Hudry, and Antoine Lobstein. Identifying and locating-dominating codes: NP-completeness results for directed graphs. *IEEE Transactions on Information Theory*, 48(8):2192–2200, 2002.

[28] Irène Charon, Olivier Hudry, and Antoine Lobstein. Minimizing the size of an identifying or locating-dominating code in a graph is NP-hard. *Theoretical Computer Science*, 290(3):2109–2120, 2003.

[29] Miroslav Chlebík and Janka Chlebíková. Approximation hardness of edge dominating set problems. *Journal of Combinatorial Optimization*, 11(3):279–290, 2006.

[30] Vašek Chvátal. A greedy heuristic for the set-covering problem. *Mathematics of Operations Research*, 4(3):233–235, 1979.

[31] Edgar Chávez, Stefan Dobrev, Evangelos Kranakis, Jaroslav Opatrny, Ladislav Stacho, and Jorge Urrutia. Local construction of planar spanners in unit disk graphs with irregular transmission ranges. In *Proc. 7th Latin American Theoretical Informatics Symposium (LATIN, Valdivia, Chile, March 2006)*, volume 3887 of *Lecture Notes in Computer Science*, pages 286–297. Springer, Berlin, Germany, 2006.

[32] Ernest J. Cockayne and Stephen T. Hedetniemi. Optimal domination in graphs. *IEEE Transactions on Circuits and Systems*, 22(11):855–857, 1975.

[33] Gérard Cohen, Iiro Honkala, Antoine Lobstein, and Gilles Zémor. On identifying codes. In *Proc. DIMACS Workshop on Codes and Association Schemes (Piscataway, NJ, USA, 1999)*, volume 56 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 97–109. American Mathematical Society, Providence, RI, USA, 2001.

[34] Charles J. Colbourn, Peter J. Slater, and Lorna K. Stewart. Locating dominating sets in series parallel networks. *Congressus Numerantium*, 56:135–162, 1987.

[35] Richard Cole and Uzi Vishkin. Deterministic coin tossing with applications to optimal parallel list ranking. *Information and Control*, 70(1):32–53, 1986.

[36] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. The MIT Press, Cambridge, MA, USA, 1990.

[37] Mary Cryan and Peter Bro Miltersen. On pseudorandom generators in NC^0. In *Proc. 26th International Symposium on Mathematical Foundations of Computer Science (MFCS, Mariánské Lázně, Czech Republic, August 2001)*, volume 2136 of *Lecture Notes in Computer Science*, pages 272–284. Springer, Berlin, Germany, 2001.

[38] David Culler, Deborah Estrin, and Mani Srivastava. Guest editors' introduction: Overview of sensor networks. *IEEE Computer*, 37(8):41–49, 2004.

[39] Andrzej Czygrinow, Michał Hańćkowiak, and Wojciech Wawrzyniak. Fast distributed approximations in planar graphs. In *Proc. 22nd International Symposium on Distributed Computing (DISC, Arcachon, France, September 2008)*, volume 5218 of *Lecture Notes in Computer Science*, pages 78–92. Springer, Berlin, Germany, 2008.

[40] Jurek Czyzowicz, Stefan Dobrev, Thomas Fevens, Hernán González-Aguilar, Evangelos Kranakis, Jaroslav Opatrny, and Jorge Urrutia. Local algorithms for dominating and connected dominating sets of unit disk graphs with location aware nodes. In *Proc. 8th Latin American Theoretical Informatics Symposium (LATIN, Búzios, Brazil, April 2008)*, volume 4957 of *Lecture Notes in Computer Science*, pages 158–169. Springer, Berlin, Germany, 2008.

[41] Jurek Czyzowicz, Stefan Dobrev, Hernán González-Aguilar, Rastislav Královič, Evangelos Kranakis, Jaroslav Opatrny, Ladislav Stacho, and Jorge Urrutia. Local 7-coloring for planar subgraphs of unit disk graphs. In *Proc. 5th International Conference on Theory and Applications of Models of Computation (TAMC, Xi'an, China, April 2008)*, volume 4978 of *Lecture Notes in Computer Science*, pages 170–181. Springer, Berlin, Germany, 2008.

[42] Jurek Czyzowicz, Stefan Dobrev, Evangelos Kranakis, Jaroslav Opatrny, and Jorge Urrutia. Local edge colouring of Yao-like subgraphs of unit disk graphs. In *Proc. 14th International Colloquium on Structural Information and Communication Complexity (SIROCCO, Castiglioncello, Italy, June 2007)*, vol-

ume 4474 of *Lecture Notes in Computer Science*, pages 195–207. Springer, Berlin, Germany, 2007.

[43] Reinhard Diestel. *Graph Theory*. Springer, Berlin, Germany, 3rd edition, 2005.

[44] Edsger W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Communications of the ACM*, 17(11):643–644, 1974.

[45] Shlomi Dolev. *Self-Stabilization*. The MIT Press, Cambridge, MA, USA, 2000.

[46] Shlomi Dolev and Nir Tzachar. Empire of colonies: Self-stabilizing and self-organizing distributed algorithms. In *Proc. 10th International Conference on Principles of Distributed Systems (OPODIS, Bordeaux, France, December 2006)*, volume 4305 of *Lecture Notes in Computer Science*, pages 230–243. Springer, Berlin, Germany, 2006.

[47] Peter G. Doyle and J. Laurie Snell. *Random Walks and Electric Networks*. Number 22 in The Carus Mathematical Monographs. The Mathematical Association of America, Washington, DC, USA, 1984.

[48] Michael Elkin. Distributed approximation: a survey. *ACM SIGACT News*, 35(4):40–57, 2004.

[49] David Eppstein, Zvi Galil, and Giuseppe F. Italiano. Dynamic graph algorithms. In Mikhail J. Atallah, editor, *Algorithms and Theory of Computation Handbook*, chapter 8. CRC Press, Boca Raton, FL, USA, 1999.

[50] Uriel Feige. A threshold of $\ln n$ for approximating set cover. *Journal of the ACM*, 45(4):634–652, 1998.

[51] Uriel Feige, Magnús M. Halldórsson, Guy Kortsarz, and Aravind Srinivasan. Approximating the domatic number. *SIAM Journal on Computing*, 32(1):172–195, 2002.

[52] Faith Fich and Eric Ruppert. Hundreds of impossibility results for distributed computing. *Distributed Computing*, 16(2–3):121–163, 2003.

[53] Patrik Floréen, Joel Kaasinen, Petteri Kaski, and Jukka Suomela. An optimal local approximation algorithm for max-min linear programs. In *Proc. 21st Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA, Calgary, Canada, August 2009)*. 2009. To appear.

[54] Patrik Floréen, Petteri Kaski, Jukka Kohonen, and Pekka Orponen. Exact and approximate balanced data gathering in energy-constrained sensor networks. *Theoretical Computer Science*, 344(1):30–46, 2005.

[55] Patrik Floréen, Petteri Kaski, Valentin Polishchuk, and Jukka Suomela. Almost stable matchings in constant time, 2008. Manuscript submitted for publication, arXiv:0812.4893 [cs.DS].

[56] Pierre Fraigniaud, Cyril Gavoille, David Ilcinkas, and Andrzej Pelc. Distributed computing with advice: Information sensitivity of graph coloring. In

Proc. 34th International Colloquium on Automata, Languages and Programming (ICALP, Wrocław, Poland, July 2007), volume 4596 of *Lecture Notes in Computer Science*, pages 231–242. Springer, Berlin, Germany, 2007.

[57] Toshihiro Fujito and Hiroshi Nagamochi. A 2-approximation algorithm for the minimum weight edge dominating set problem. *Discrete Applied Mathematics*, 118(3):199–207, 2002.

[58] K. Ruben Gabriel and Robert R. Sokal. A new statistical approach to geographic variation analysis. *Systematic Zoology*, 18(3):259–278, 1969.

[59] David Gale and Lloyd S. Shapley. College admissions and the stability of marriage. *The American Mathematical Monthly*, 69(1):9–15, 1962.

[60] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, New York, NY, USA, 1979.

[61] Laszek Gąsieniec, Chang Su, and Prudence Wong. Routing in geometric networks. In Ming-Yang Kao, editor, *Encyclopedia of Algorithms*. Springer, New York, NY, USA, 2008.

[62] Phillip B. Gibbons. Fun with networks: social, sensor, and shape shifting. Invited talk, 22nd International Symposium on Distributed Computing (DISC, Arcachon, France), September 2008.

[63] Chris Godsil and Gordon Royle. *Algebraic Graph Theory*, volume 207 of *Graduate Texts in Mathematics*. Springer, New York, NY, USA, 2004.

[64] Andrew V. Goldberg, Serge A. Plotkin, and Gregory E. Shannon. Parallel symmetry-breaking in sparse graphs. *SIAM Journal on Discrete Mathematics*, 1(4):434–446, 1988.

[65] Ronald L. Graham, Bruce L. Rothschild, and Joel H. Spencer. *Ramsey Theory*. John Wiley & Sons, New York, NY, USA, 1980.

[66] Sylvain Gravier, Ralf Klasing, and Julien Moncel. Hardness results and approximation algorithms for identifying codes and locating-dominating codes in graphs. Technical Report RR-1417-06, Laboratoire Bordelais de Recherche en Informatique (LaBRI), Talence, France, November 2006.

[67] Sylvain Gravier, Ralf Klasing, and Julien Moncel. Hardness results and approximation algorithms for identifying codes and locating-dominating codes in graphs. *Algorithmic Operations Research*, 3(1):43–50, 2008.

[68] Dan Gusfield and Robert W. Irving. *The Stable Marriage Problem: Structure and Algorithms*. The MIT Press, Cambridge, MA, USA, 1989.

[69] Bruce Hajek and Galen Sasaki. Link scheduling in polynomial time. *IEEE Transactions on Information Theory*, 34(5):910–917, 1988.

[70] Michał Hańćkowiak, Michał Karoński, and Alessandro Panconesi. On the distributed complexity of computing maximal matchings. In *Proc. 9th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA, San Francisco,*

CA, USA, January 1998), pages 219–225. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1998.

[71] Michał Hańćkowiak, Michał Karoński, and Alessandro Panconesi. On the distributed complexity of computing maximal matchings. *SIAM Journal on Discrete Mathematics*, 15(1):41–57, 2001.

[72] Marja Hassinen, Evangelos Kranakis, Valentin Polishchuk, Jukka Suomela, and Andreas Wiese. Analysing local algorithms in location-aware quasi unit-disk graphs, 2009. Manuscript submitted for publication.

[73] Marja Hassinen, Valentin Polishchuk, and Jukka Suomela. Local 3-approximation algorithms for weighted dominating set and vertex cover in quasi unit-disk graphs. In *Proc. 2nd International Workshop on Localized Algorithms and Protocols for Wireless Sensor Networks (LOCALGOS, Santorini Island, Greece, June 2008)*, pages V.9–V.12. 2008.

[74] Dorit S. Hochbaum. Approximation algorithms for the set covering and vertex cover problems. *SIAM Journal on Computing*, 11(3):555–556, 1982.

[75] Dorit S. Hochbaum and Wolfgang Maass. Approximation schemes for covering and packing problems in image processing and VLSI. *Journal of the ACM*, 32(1):130–136, 1985.

[76] John G. Hocking and Gail S. Young. *Topology*. Addison-Wesley, Reading, MA, USA, 1961.

[77] Jaap-Henk Hoepman, Shay Kutten, and Zvi Lotker. Efficient distributed weighted matchings on trees. In *Proc. 13th International Colloquium on Structural Information and Communication Complexity (SIROCCO, Chester, UK, July 2006)*, volume 4056 of *Lecture Notes in Computer Science*, pages 115–129. Springer, Berlin, Germany, 2006.

[78] Bo Hong and Viktor K. Prasanna. Constrained flow optimization with applications to data gathering in sensor networks. In *Proc. 1st International Workshop on Algorithmic Aspects of Wireless Sensor Networks (Algosensors, Turku, Finland, July 2004)*, volume 3121 of *Lecture Notes in Computer Science*, pages 187–200. Springer, Berlin, Germany, 2004.

[79] Shlomo Hoory. *On Graphs of High Girth*. PhD thesis, Hebrew University, Jerusalem, March 2002.

[80] Harry B. Hunt, III, Madhav V. Marathe, Venkatesh Radhakrishnan, S. S. Ravi, Daniel J. Rosenkrantz, and Richard E. Stearns. NC-approximation schemes for NP- and PSPACE-hard problems for geometric graphs. *Journal of Algorithms*, 26(2):238–274, 1998.

[81] Johan Håstad. One-way permutations in NC^0. *Information Processing Letters*, 26(3):153–155, 1987.

[82] Johan Håstad. Clique is hard to approximate within $n^{1-\epsilon}$. *Acta Mathematica*, 182(1):105–142, 1999.

[83] Wilfried Imrich and Tomaž Pisanski. Multiple Kronecker covering graphs. *European Journal of Combinatorics*, 29(5):1116–1122, 2008.

[84] Kamal Jain, Jitendra Padhye, Venkata N. Padmanabhan, and Lili Qiu. Impact of interference on multi-hop wireless network performance. *Wireless Networks*, 11(4):471–487, 2005.

[85] Klaus Jansen. Approximate strong separation with application in fractional graph coloring and preemptive scheduling. *Theoretical Computer Science*, 302(1–3):239–256, 2003.

[86] Jerzy W. Jaromczyk and Godfried T. Toussaint. Relative neighborhood graphs and their relatives. *Proceedings of the IEEE*, 80(9):1502–1517, 1992.

[87] David S. Johnson. Approximation algorithms for combinatorial problems. *Journal of Computer and System Sciences*, 9:256–278, 1974.

[88] Ralph E. Johnson and Fred B. Schneider. Symmetry and similarity in distributed systems. In *Proc. 4th Annual ACM Symposium on Principles of Distributed Computing (PODC, Minaki, Ontario, Canada, August 1985)*, pages 13–22. ACM Press, New York, NY, USA, 1985.

[89] Konstantinos Kalpakis, Koustuv Dasgupta, and Parag Namjoshi. Efficient algorithms for maximum lifetime data gathering and aggregation in wireless sensor networks. *Computer Networks*, 42(6):697–716, 2003.

[90] Iyad A. Kanj, Ljubomir Perković, and Ge Xia. Computing lightweight spanners locally. In *Proc. 22nd International Symposium on Distributed Computing (DISC, Arcachon, France, September 2008)*, volume 5218 of *Lecture Notes in Computer Science*, pages 365–378. Springer, Berlin, Germany, 2008.

[91] Mark G. Karpovsky, Krishnendu Chakrabarty, and Lev B. Levitin. On a new class of codes for identifying vertices in graphs. *IEEE Transactions on Information Theory*, 44(2):599–611, 1998.

[92] J. Mark Keil and Carl A. Gutwin. Classes of graphs which approximate the complete Euclidean graph. *Discrete & Computational Geometry*, 7(1):13–28, 1992.

[93] Sanjeev Khanna, Rajeev Motwani, Madhu Sudan, and Umesh Vazirani. On syntactic versus computational views of approximability. *SIAM Journal on Computing*, 28(1):164–191, 1999.

[94] Subhash Khot. Improved inapproximability results for MaxClique, chromatic number and approximate graph coloring. In *Proc. 42nd Annual Symposium on Foundations of Computer Science (FOCS, Las Vegas, NV, USA, October 2001)*, pages 600–609. IEEE Computer Society Press, Los Alamitos, CA, USA, 2001.

[95] Samir Khuller, Balaji Raghavachari, and Azriel Rosenfeld. Landmarks in graphs. *Discrete Applied Mathematics*, 70(3):217–229, 1996.

[96] Amos Korman and Shay Kutten. On distributed verification. In *Proc. 8th International Conference on Distributed Computing and Networking (ICDCN,*

Guwahati, India, December 2006), volume 4308 of *Lecture Notes in Computer Science*, pages 100–114. Springer, Berlin, Germany, 2006.

[97] Amos Korman, Shay Kutten, and David Peleg. Proof labeling schemes. In *Proc. 24th Annual ACM Symposium on Principles of Distributed Computing (PODC, Las Vegas, NV, USA, July 2005)*, pages 9–18. ACM Press, New York, NY, USA, 2005.

[98] Bernhard Korte and Jens Vygen. *Combinatorial Optimization: Theory and Algorithms*. Springer, Berlin, Germany, 3rd edition, 2006.

[99] Farinaz Koushanfar, Nina Taft, and Miodrag Potkonjak. Sleeping coordination for comprehensive sensing using isotonic regression and domatic partitions. In *Proc. 25th Conference on Computer Communications (INFOCOM, Barcelona, Spain, April 2006)*. IEEE, Piscataway, NJ, USA, 2006.

[100] Evangelos Kranakis, Harvinder Singh, and Jorge Urrutia. Compass routing on geometric networks. In *Proc. 11th Canadian Conference on Computational Geometry (CCCG, Vancouver, BC, Canada, August 1999)*. 1999.

[101] Bhaskar Krishnamachari. *Networking Wireless Sensors*. Cambridge University Press, Cambridge, UK, 2005.

[102] Fabian Kuhn. *The Price of Locality: Exploring the Complexity of Distributed Coordination Primitives*. PhD thesis, ETH Zürich, 2005.

[103] Fabian Kuhn. Local approximation of covering and packing problems. In Ming-Yang Kao, editor, *Encyclopedia of Algorithms*. Springer, New York, NY, USA, 2008.

[104] Fabian Kuhn and Thomas Moscibroda. Distributed approximation of capacitated dominating sets. In *Proc. 19th Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA, San Diego, CA, USA, June 2007)*, pages 161–170. ACM Press, New York, NY, USA, 2007.

[105] Fabian Kuhn, Thomas Moscibroda, and Roger Wattenhofer. What cannot be computed locally! In *Proc. 23rd Annual ACM Symposium on Principles of Distributed Computing (PODC, St. John's, Newfoundland, Canada, July 2004)*, pages 300–309. ACM Press, New York, NY, USA, 2004.

[106] Fabian Kuhn, Thomas Moscibroda, and Roger Wattenhofer. On the locality of bounded growth. In *Proc. 24th Annual ACM Symposium on Principles of Distributed Computing (PODC, Las Vegas, NV, USA, July 2005)*, pages 60–68. ACM Press, New York, NY, USA, 2005.

[107] Fabian Kuhn, Thomas Moscibroda, and Roger Wattenhofer. Fault-tolerant clustering in ad hoc and sensor networks. In *Proc. 26th IEEE International Conference on Distributed Computing Systems (ICDCS, Lisboa, Portugal, July 2006)*. IEEE Computer Society Press, Los Alamitos, CA, USA, 2006.

[108] Fabian Kuhn, Thomas Moscibroda, and Roger Wattenhofer. The price of being near-sighted. In *Proc. 17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA, Miami, FL, USA, January 2006)*, pages 980–989. ACM Press, New York, NY, USA, 2006.

[109] Fabian Kuhn, Thomas Moscibroda, and Roger Wattenhofer. The price of being near-sighted. Technical Report 229, ETH Zürich, Computer Engineering and Networks Laboratory, January 2006.

[110] Fabian Kuhn and Roger Wattenhofer. Constant-time distributed dominating set approximation. *Distributed Computing*, 17(4):303–310, 2005.

[111] Fabian Kuhn and Roger Wattenhofer. On the complexity of distributed graph coloring. In *Proc. 25th Annual ACM Symposium on Principles of Distributed Computing (PODC, Denver, CO, USA, July 2006)*, pages 7–15. ACM Press, New York, NY, USA, 2006.

[112] Fabian Kuhn, Roger Wattenhofer, and Aaron Zollinger. Ad hoc networks beyond unit disk graphs. *Wireless Networks*, 14(5):715–729, 2008.

[113] Moshe Laifenfeld. Localization and identification in networks using robust identifying codes. In *Proc. 2008 Information Theory and Applications Workshop (San Diego, CA, USA, January–February 2008)*, pages 165–174. IEEE, Piscataway, NJ, USA, 2008.

[114] Moshe Laifenfeld and Ari Trachtenberg. Identifying codes and covering problems. *IEEE Transactions on Information Theory*, 54(9):3929–3950, 2008.

[115] Moshe Laifenfeld, Ari Trachtenberg, and Tanya Berger-Wolf. Identifying codes and the set cover problem. In *Proc. 44th Annual Allerton Conference on Communication, Control, and Computing (Monticello, IL, USA, September 2006)*. 2006.

[116] Kadathur B. Lakshmanan, Daniel J. Rosenkrantz, and S. S. Ravi. Alarm placement in systems with fault propagation. *Theoretical Computer Science*, 243(1–2):269–288, 2000.

[117] Felix Lazebnik and Vasiliy A. Ustimenko. Explicit construction of graphs with an arbitrary large girth and of large size. *Discrete Applied Mathematics*, 60(1–3):275–284, 1995.

[118] Christoph Lenzen, Yvonne Anne Oswald, and Roger Wattenhofer. What can be approximated locally? In *Proc. 20th Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA, Munich, Germany, June 2008)*, pages 46–54. ACM Press, New York, NY, USA, 2008.

[119] Christoph Lenzen and Roger Wattenhofer. Leveraging Linial's locality limit. In *Proc. 22nd International Symposium on Distributed Computing (DISC, Arcachon, France, September 2008)*, volume 5218 of *Lecture Notes in Computer Science*, pages 394–407. Springer, Berlin, Germany, 2008.

[120] Xiang-Yang Li, Yu Wang, and Wen-Zhan Song. Applications of k-local MST for topology control and broadcasting in wireless ad hoc networks. *IEEE Transactions on Parallel and Distributed Systems*, 15(12):1057–1069, 2004.

[121] Nathan Linial. Locality in distributed graph algorithms. *SIAM Journal on Computing*, 21(1):193–201, 1992.

[122] Antoine Lobstein. Codes identifiants, localisateurs-dominateurs et discrim-

inants dans les graphes, January 2009. http://www.infres.enst.fr/~lobstein/bibLOCDOMetID.html.

[123] László Lovász. On the ratio of optimal integral and fractional covers. *Discrete Mathematics*, 13(4):383–390, 1975.

[124] László Lovász. Very large graphs, December 2008. Manuscript, arXiv: 0902.0132 [math.CO].

[125] Alexander Lubotzky, Ralph Phillips, and Peter Sarnak. Ramanujan graphs. *Combinatorica*, 8(3):261–277, 1988.

[126] Michael Luby. Removing randomness in parallel computation without a processor penalty. *Journal of Computer and System Sciences*, 47(2):250–286, 1993.

[127] Michael Luby and Noam Nisan. A parallel approximation algorithm for positive linear programming. In *Proc. 25th Annual ACM Symposium on Theory of Computing (STOC, San Diego, CA, USA, May 1993)*, pages 448–457. ACM Press, New York, NY, USA, 1993.

[128] Carsten Lund and Mihalis Yannakakis. On the hardness of approximating minimization problems. *Journal of the ACM*, 41(5):960–981, 1994.

[129] Nancy A. Lynch. A hundred impossibility proofs for distributed computing. In *Proc. 8th Annual ACM Symposium on Principles of Distributed Computing (PODC, Edmonton, Canada, August 1989)*, pages 1–28. ACM Press, New York, NY, USA, 1989.

[130] Alain Mayer, Moni Naor, and Larry Stockmeryer. Local computations on static and dynamic graphs. In *Proc. 3rd Israel Symposium on the Theory of Computing and Systems (ISTCS, Tel Aviv, Israel, January 1995)*, pages 268–278. IEEE, Piscataway, NJ, USA, 1995.

[131] Brendan D. McKay, Nicholas C. Wormald, and Beata Wysocka. Short cycles in random regular graphs. *Electronic Journal of Combinatorics*, 11(1):#R66, 2004.

[132] Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, Cambridge, UK, 2005.

[133] Julien Moncel. Optimal graphs for identification of vertices in networks. Technical Report 138, Laboratoire Leibniz, Grenoble, France, November 2005.

[134] Thomas Moscibroda. *Locality, Scheduling, and Selfishness: Algorithmic Foundations of Highly Decentralized Networks*. PhD thesis, ETH Zürich, 2006.

[135] Thomas Moscibroda and Roger Wattenhofer. Maximizing the lifetime of dominating sets. In *Proc. 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS, Denver, CO, USA, April 2005)*, page 242b. IEEE Computer Society Press, Los Alamitos, CA, USA, 2005.

[136] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, Cambridge, UK, 1995.

[137] James R. Munkres. *Topology*. Prentice Hall, Upper Saddle River, NJ, USA, 2nd edition, 2000.

[138] Moni Naor and Larry Stockmeyer. What can be computed locally? *SIAM Journal on Computing*, 24(6):1259–1277, 1995.

[139] Huy N. Nguyen and Krzysztof Onak. Constant-time approximation algorithms via local improvements. In *Proc. 49th Annual IEEE Symposium on Foundations of Computer Science (FOCS, Philadelphia, PA, USA, October 2008)*, pages 327–336. IEEE Computer Society Press, Los Alamitos, CA, USA, 2008.

[140] Christos H. Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Dover Publications, Inc., Mineola, NY, USA, 1998.

[141] Christos H. Papadimitriou and Mihalis Yannakakis. On the value of information in distributed decision-making. In *Proc. 10th Annual ACM Symposium on Principles of Distributed Computing (PODC, Montreal, Quebec, Canada, August 1991)*, pages 61–64. ACM Press, New York, NY, USA, 1991.

[142] Christos H. Papadimitriou and Mihalis Yannakakis. Optimization, approximation, and complexity classes. *Journal of Computer and System Sciences*, 43(3):425–440, 1991.

[143] Christos H. Papadimitriou and Mihalis Yannakakis. Linear programming without the matrix. In *Proc. 25th Annual ACM Symposium on Theory of Computing (STOC, San Diego, CA, USA, May 1993)*, pages 121–129. ACM Press, New York, NY, USA, 1993.

[144] Michal Parnas and Dana Ron. Approximating the minimum vertex cover in sublinear time and a connection to distributed algorithms. *Theoretical Computer Science*, 381(1–3):183–196, 2007.

[145] David Peleg. *Distributed Computing – A Locality-Sensitive Approachconwa*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000.

[146] Sriram V. Pemmaraju and Imran A. Pirwani. Energy conservation via domatic partitions. In *Proc. 7th ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc, Florence, Italy, May 2006)*, pages 143–154. ACM Press, New York, NY, USA, 2006.

[147] Valentin Polishchuk and Jukka Suomela. A simple local 3-approximation algorithm for vertex cover. *Information Processing Letters*, 109(12):642–645, 2009.

[148] Frank P. Ramsey. On a problem of formal logic. *Proceedings of the London Mathematical Society*, 30:264–286, 1930.

[149] Nageswara S. V. Rao. Computational complexity issues in operative diagnos-

tics of graph-based systems. *IEEE Transactions on Computers*, 42(4):447–457, 1993.

[150] Saikat Ray, David Starobinski, Ari Trachtenberg, and Rachanee Ungrangsi. Robust location detection with sensor networks. *IEEE Journal on Selected Areas in Communications*, 22(6):1016–1025, 2004.

[151] Saikat Ray, Rachanee Ungrangsi, Francesco De Pellegrini, Ari Trachtenberg, and David Starobinski. Robust location detection in emergency sensor networks. In *Proc. 22nd Conference on Computer Communications (INFOCOM, San Francisco, CA, USA, March–April 2003)*, pages 1044–1053. IEEE, Piscataway, NJ, USA, 2003.

[152] Ran Raz and Shmuel Safra. A sub-constant error-probability low-degree test, and a sub-constant error-probability PCP characterization of NP. In *Proc. 29th Annual ACM Symposium on Theory of Computing (STOC, El Paso, TX, USA, May 1997)*, pages 475–484. ACM Press, New York, NY, USA, 1997.

[153] Narayanan Sadagopan and Bhaskar Krishnamachari. Maximizing data extraction in energy-limited sensor networks. In *Proc. 23rd Conference on Computer Communications (INFOCOM, Hong Kong, March 2004)*, pages 1717–1727. IEEE, Piscataway, NJ, USA, 2004.

[154] Johannes Schneider and Roger Wattenhofer. A log-star distributed maximal independent set algorithm for growth-bounded graphs. In *Proc. 27th Annual ACM Symposium on Principles of Distributed Computing (PODC, Toronto, Canada, August 2008)*, pages 35–44. ACM Press, New York, NY, USA, 2008.

[155] Marco Schneider. Self-stabilization. *ACM Computing Surveys*, 25(1):45–67, 1993.

[156] Peter J. Slater. Domination and location in acyclic graphs. *Networks*, 17(1):55–64, 1987.

[157] Petra Šparl and Janez Žerovnik. 2-local 4/3-competitive algorithm for multicoloring hexagonal graphs. *Journal of Algorithms*, 55(1):29–41, 2005.

[158] Aaron D. Sterling. A limit to the power of multiple nucleation in self-assembly. In *Proc. 22nd International Symposium on Distributed Computing (DISC, Arcachon, France, September 2008)*, volume 5218 of *Lecture Notes in Computer Science*, pages 451–465. Springer, Berlin, Germany, 2008.

[159] Jukka Suomela. Survey of local algorithms. `http://www.iki.fi/jukka.suomela/local-survey`, 2009. Manuscript submitted for publication.

[160] Godfried T. Toussaint. The relative neighbourhood graph of a finite planar set. *Pattern Recognition*, 12(4):261–268, 1980.

[161] Jorge Urrutia. Local solutions for global problems in wireless networks. *Journal of Discrete Algorithms*, 5(3):395–407, 2007.

[162] Vijay V. Vazirani. *Approximation Algorithms*. Springer, Berlin, Germany, 2001.

[163] Yu Wang and Xiang-Yang Li. Localized construction of bounded degree and planar spanner for wireless ad hoc networks. *Mobile Networks and Applications*, 11(2):161–175, 2006.

[164] Mirjam Wattenhofer and Roger Wattenhofer. Distributed weighted matching. In *Proc. 18th International Symposium on Distributed Computing (DISC, Amsterdam, Netherlands, October 2004)*, volume 3274 of *Lecture Notes in Computer Science*, pages 335–348. Springer, Berlin, Germany, 2004.

[165] Roger Wattenhofer and Aaron Zollinger. XTC: a practical topology control algorithm for ad-hoc networks. In *Proc. 18th IEEE International Parallel and Distributed Processing Symposium (IPDPS, Santa Fe, NM, USA, April 2004)*. IEEE Computer Society Press, Los Alamitos, CA, USA, 2004.

[166] Paul M. Weichsel. The Kronecker product of graphs. *Proceedings of the American Mathematical Society*, 13(1):47–52, 1962.

[167] Matt Welsh. Sensor networks for the sciences: lessons from the field. Invited talk, 4th IEEE International Conference on Distributed Computing in Sensor Systems (DCOSS, Santorini Island, Greece), June 2008.

[168] Andreas Wiese. Local approximation algorithms in unit disk graphs. Master's thesis, Technische Universität Berlin, 2007.

[169] Andreas Wiese and Evangelos Kranakis. Impact of locality on location aware unit disk graphs. *Algorithms*, 1:2–29, 2008.

[170] Andreas Wiese and Evangelos Kranakis. Local construction and coloring of spanners of location aware unit disk graphs. In *Proc. 34th International Workshop on Graph-Theoretic Concepts in Computer Science (WG, Durham, UK, June–July 2008)*, volume 5344 of *Lecture Notes in Computer Science*, pages 372–383. Springer, Berlin, Germany, 2008.

[171] Andreas Wiese and Evangelos Kranakis. Local maximal matching and local 2-approximation for vertex cover in UDGs. In *Proc. 7th International Conference on Ad-Hoc Networks & Wireless (AdHoc-NOW, Sophia Antipolis, France, September 2008)*, volume 5198 of *Lecture Notes in Computer Science*, pages 1–14. Springer, Berlin, Germany, 2008.

[172] Andreas Wiese and Evangelos Kranakis. Local PTAS for independent set and vertex cover in location aware unit disk graphs. In *Proc. 4th IEEE International Conference on Distributed Computing in Sensor Systems (DCOSS, Santorini Island, Greece, June 2008)*, volume 5067 of *Lecture Notes in Computer Science*, pages 415–431. Springer, Berlin, Germany, 2008.

[173] Andreas Wiese and Evangelos Kranakis. Local PTAS for dominating and connected dominating set in location aware unit disk graphs. In *Proc. 6th Workshop on Approximation and Online Algorithms (WAOA, Karlsruhe, Germany, September 2008)*, volume 5426 of *Lecture Notes in Computer Science*, pages 227–240. Springer, Berlin, Germany, 2009.

[174] Masafumi Yamashita and Tsunehiko Kameda. Computing on anonymous

networks: Part I – characterizing the solvable cases. *IEEE Transactions on Parallel and Distributed Systems*, 7(1):69–89, 1996.

[175] Mihalis Yannakakis and Fanica Gavril. Edge dominating sets in graphs. *SIAM Journal on Applied Mathematics*, 38(3):364–372, 1980.

[176] Andrew Chi-Chih Yao. On constructing minimum spanning trees in k-dimensional spaces and related problems. *SIAM Journal on Computing*, 11(4):721–736, 1982.

[177] Jennifer Yick, Biswanath Mukherjee, and Dipak Ghosal. Wireless sensor network survey. *Computer Networks*, 52(12):2292–2330, 2008.

[178] Neal E. Young. Sequential and parallel algorithms for mixed packing and covering. In *Proc. 42nd Annual Symposium on Foundations of Computer Science (FOCS, Las Vegas, NV, USA, October 2001)*, pages 538–546. IEEE Computer Society Press, Los Alamitos, CA, USA, 2001.

[179] Aaron Zollinger. Geographic routing. In Ming-Yang Kao, editor, *Encyclopedia of Algorithms*. Springer, New York, NY, USA, 2008.

Index