Cellular Distributed and Parallel Computing

Lei Xu

St. Anne's College

University of Oxford

A thesis submitted for the degree of

Doctor of Philosophy

Hilary Term 2014

Cellular Distributed and Parallel Computing

Lei Xu
St. Anne's College, Oxford
Doctor of Philosophy, Hilary Term 2014

Abstract

This thesis focuses on novel approaches to distributed and parallel computing that are inspired by the mechanism and functioning of biological cells. We refer to this concept as *cellular distributed and parallel computing* which focuses on three important principles: *simplicity*, *parallelism*, and *locality*.

We first give a parallel polynomial-time solution to the constraint satisfaction problem (CSP) based on a theoretical model of cellular distributed and parallel computing, which is known as neural-like P systems (or neural-like membrane systems).

We then design a class of simple neural-like P systems to solve the fundamental maximal independent set (MIS) selection problem efficiently in a distributed way, by drawing inspiration from the way that developing cells in the fruit fly become specialised.

Building on the novel bio-inspired approach to distributed MIS selection, we propose a new simple randomised algorithm for another fundamental distributed computing problem: the distributed greedy colouring (GC) problem.

We then propose an improved distributed MIS selection algorithm that incorporates for the first time another important feature of the biological system: adapting the probabilities used at each node based on local feedback from neighbouring nodes. The improved distributed MIS selection algorithm is again extended to solve the distributed greedy colouring problem. Both improved algorithms are simple and robust and work under very restrictive conditions, moreover, they both achieve state-of-the-art performance in terms of their worst-case time complexity and message complexity. Given any n-node graph with maximum degree Δ, the expected time complexity of our improved distributed MIS selection algorithm is $O(\log n)$ and the message complexity per node is $O(1)$. The expected time complexity of our improved distributed greedy colouring algorithm is $O(\Delta + \log n)$ and the message complexity per node is again $O(1)$.

Finally, we provide some experimental results to illustrate the time and message complexity of our proposed algorithms in practice. In particular, we show experimentally that the number of colours used by our distributed greedy colouring algorithms turns out to be optimal or near-optimal for many standard graph colouring benchmarks, so they provide effective simple heuristic approaches to computing a colouring with a small number of colours.

Acknowledgements

Completing my doctoral study at Oxford is probably one of my most memorable experiences. I really appreciate the joy that Oxford has brought to me in both St. Anne's College and the Department of Computer Science. Oxford is not only like a sacred castle that has shielded me from the crowd and hullabaloo outside, but also like a gracious old person who is patiently telling stories of long long time ago to countless fascinated children.

I am grateful to many people during my study at Oxford. My first debt of gratitude must go to my supervisor, Professor Peter Jeavons. Pete gave me great freedom to pursue independent work, patiently guided my way towards successful research, offered me a lot of very useful help, and always encouraged me throughout my study. Doing research with him is really an enjoyable thing which makes my study at Oxford more valuable and memorable. Besides, Pete also taught and inspired me a lot of wisdom about life. It's one of my greatest luck in my life that I had a supervisor like Pete.

I also give my sincere thanks to Professor Alex Scott, who has helped me out when I was stuck with my research in the second year. I could not complete my work without his insightful assistance. I am also very grateful to my colleagues at Oxford for their support: David Cohen, Stanislav Živný, Justyna Petke, Evgenij Thorstensen, Markus Aschinger, András Salamon, Michael Morak, Helen Flynn, and Victor Spirin.

My friends from the department and from St. Anne's College also deserve my sincere thanks. You were my source of laughter, relaxation and happiness. Thank you for bringing me so much happy time at Oxford.

I also give my special thanks to those who are not in Oxford but gave me unconditional support and encouragement directly or indirectly. You are always like a beamer from far away guiding me to keep calm and carry on.

Finally, I would like to dedicate this work to my parents: Mr. Kaishan Xu and Mrs. Zhiqin Zhang. I hope that my work at Oxford makes you proud. I love you and owe you my heartfelt appreciation.

Declarations

I hereby declare that I have written this thesis entirely by myself. Parts of the thesis have appeared in the following papers which have been subject to peer review.

- Peter Jeavons, Alex Scott, **Lei Xu**, Feedback from Nature: Randomised Distributed Algorithms for Maximal Independent Set Selection and Greedy Colouring, *in submission*.
- **Lei Xu**, Peter Jeavons, Patterns from Nature: Distributed Greedy Colouring with Simple Messages and Minimal Graph Knowledge, *in submission*.
- **Lei Xu**, Xiangxiang Zeng, Peter Jeavons, A Polynomial Time Solution to Constraint Satisfaction Problems by Neural-like P Systems, *International Journal of Unconventional Computing* (**IJUC**) 9 (5-6) (2013) 465-481.
- **Lei Xu**, Peter Jeavons, Simple Neural-like P Systems for Maximal Independent Set Selection, *Neural Computation* (**NECO**) 25 (6) (2013) 1642-1659.
- Alex Scott, Peter Jeavons, **Lei Xu**, Feedback from Nature: An Optimal Distributed Algorithm for Maximal Independent Set Selection, *In Proceedings of the 2013 ACM symposium on principles of distributed computing (**PODC'13**). ACM, New York, NY, USA* (2013) 147-156.
- **Lei Xu**, Xiangxiang Zeng, Peter Jeavons, A Polynomial Time Solution to Constraint Satisfaction Problems by Neural-like P Systems, *In Asian Conference on Membrane Computing 2012 (**ACMC'12**), Wuhan, China*.

Some of the results in these papers were obtained in collaboration. I would like to express here my thanks to my coauthors: Peter Jeavons, Alex Scott and Xiangxiang Zeng.

Contents

List of Figures

CHAPTER 1

Introduction

Biological cells, which can be seen as the basic information processing units of life, seem to have great power to tackle complicated tasks by using only simple operations. Since biological cells are inherently distributed and parallel, this power is particularly relevant in the area of *distributed and parallel computing*: the challenges of deployment and communication between networked computers highly resemble those between biological cells in a multicellular organism, however, biological cells can often complete tasks in a very simple, efficient and robust way.

This thesis focuses on novel approaches to distributed and parallel computing that are inspired by the mechanism and functioning of biological cells. We refer to this concept as *cellular distributed and parallel computing* (in short, *cellular computing*) which focuses on three important principles: *simplicity*, *parallelism*, and *locality*.

In Chapter 2, we present an overview of cellular computing including particularly a theoretical cellular computing model: *P systems*. In addition, we present an intensive review of some classical computational problems to be tackled by cellular computing in the thesis: the *constraint satisfaction problem* (CSP), the distributed *maximal independent set* (MIS) selection problem and the distributed *greedy colouring* (GC) problem.

Membrane systems (also known as P systems) have been introduced as a class of parallel, non-deterministic, synchronous and distributed models of computation inspired by the structure and functioning of living cells [PRS10]. As a model of cellular computing, P systems have been applied in various research areas, ranging from biology to linguistics and computer science [RCPJ08, CdAPH$^+$10, CCM$^+$11, Xu12]. In particular, a large variety of P systems have been proposed to solve **NP**-hard problems in polynomial time. These solutions are usually obtained by generating an

exponential amount of workspace in polynomial time and using parallelism to check simultaneously all the candidate solutions [LZFM07, DPGNPJRNn08, XLZ10, PPPJ11].

Constraints are an ubiquitous concept [RBW06]. A large number of problems in artificial intelligence and other areas of computer science can be viewed as special cases of the constraint satisfaction problem. Each instance of the CSP can be described as follows: a set of variables, a finite and discrete domain for each variable, and a set of constraints are given. Each constraint is defined over some subset of the original set of variables and limits the combinations of values that the variables in this subset can take. To solve the CSP is to find an assignment of values to all of the variables that satisfies all the constraints. Alternatively, in some instances, to solve the CSP is to find all such assignments. Generally, solving the CSP is **NP**-hard [RBW06].

In Chapter 3, we design neural-like P systems for solving the CSP. This is the first proposed use of P systems to solve the general CSP. Moreover, the method we propose does not rely on generating exponential workspace in polynomial time as in the conventional methods of using P systems to solve **NP**-hard problems. We show that by broadcasting symbol-impulses to neighbouring cells and processing multisets or strings of symbol-impulses from neighbouring cells, neural-like P systems can in principle solve the CSP in polynomial time without using cell division or cell separation. The related results of Chapter 3 were published in [XZJ13].

In distributed computing, a large number of processors jointly and distributively achieve a computing task, without any one processor receiving all the inputs or observing all the outputs. There is a trend in distributed computing towards algorithms that can operate efficiently under very restrictive conditions [AAB$^+$11, EW13]. A prominent example of this trend is a novel randomised approach to distributed maximal independent set selection inspired by the study of the way that developing cells in the fruit fly become specialised [AAB$^+$11, AABJ$^+$11].

A fundamental problem in distributed computing is that of electing a set of local leaders in a network of connected processors, known as the maximal independent set selection problem. An MIS is a maximal set of nodes in a network such that no two of them are neighbours. The selection of an MIS in a given graph is a very well-studied graph-algorithmic problem; it also serves as a basic building block in many other distributed algorithms [ABI86, Lub85, KMNW05]. Distributively selecting an MIS has been considered a challenging problem for three decades [AAB$^+$11] and has been extensively studied in various models [Lub85, Lin92, ABI86, KMNW05, KMW06, MW05].

In Chapter 4, we design a class of simple neural-like P systems—neural-like probabilistic P systems—which implements a distributed MIS selection algorithm that was originally inspired by the fly's nervous system [AAB$^+$11, AABJ$^+$11]. Hence we implement a biologically-inspired algorithm using a class of biologically-inspired computing models. This new class of P systems possesses two features that are attractive for both distributed computing and membrane computing:

first, the individual processors do not need any information about the graph; second, they communicate using only one-bit messages. Our simple neural-like P systems for performing the MIS selection algorithm give further insight into the possible mechanisms employed by biological systems. These models also illustrate a new method for using membrane computing, and provide a new bridge between membrane computing and distributed computing. The related results of Chapter 4 were published in [XJ13].

Another fundamental problem in distributed computing that is closely related to the distributed MIS selection problem is the $(\Delta + 1)$-*colouring* problem. In this problem the aim is to colour the vertices of a graph which has maximum degree Δ using no more than $\Delta + 1$ colours so that adjacent vertices are assigned different colours. Like the distributed MIS selection problem, the distributed $(\Delta + 1)$-colouring problem also serves as a basic building block in many other distributed algorithms, and has many applications for resource assignment, in particular for frequency assignment in radio-communication networks [MP12, PL96, Wat05, NSW11]. Because of this, it has also been extensively studied [BE09, BE11, Kuh09, PS96, SW10, OJ99, PR01, HJS03, HKKN04].

A more restricted variant of the colouring problem is called greedy colouring [Gru39, GKK$^+$09], where the aim is to obtain a colouring with the property that no individual vertex can be recoloured with a smaller colour (in some fixed ordering of the colours). Computing a greedy colouring distributively is believed to be more difficult than computing an arbitrary $(\Delta + 1)$-colouring distributively [GKK$^+$09], but such colourings often use a much smaller number of colours.

In Chapter 5, building on the novel bio-inspired approach to distributed MIS selection, we propose a new simple randomised algorithm for distributed greedy colouring. In our approach the processors exchange only simple messages representing colours, have minimal graph knowledge and cannot distinguish between their neighbours. Two variations of this algorithm are investigated and compared. The related results of Chapter 5 are summarized in [XJ14] which has been submitted for publication.

In Chapter 6, we consider improving on the distributed MIS selection algorithm in Chapter 4 and the distributed GC algorithm in Chapter 5, by incorporating for the first time another important feature of the biological system: adapting the probabilities used at each node based on local feedback from neighbouring nodes.

We first propose an improved randomised distributed MIS selection algorithm. Our new algorithm is simple and robust and is able to operate under very restrictive conditions. The algorithm assumes an identical anonymous processor at each node that has *no information about the network*. At each time step, each node can only *broadcast a single bit* to all its neighbours, or remain silent. Each node can detect whether one or more neighbours have broadcast, but cannot tell how many neighbours have broadcast, *or which ones*.

We prove that our algorithm is optimal in both worst-case time and message complexity under such conditions, by showing that it runs in expected $O(\log n)$ time, where n is the number of nodes, and the expected number of messages sent by each node is bounded by a constant, regardless of the network.

We then extend the approach to obtain an algorithm for the distributed greedy colouring problem. This algorithm also runs under very restrictive conditions where the processors are anonymous and have no information about the network. For this problem we allow each node to broadcast only a single message to all neighbours at each time step representing a single desired colour value. Once again nodes can only detect whether at least one neighbour has broadcast a colour, and cannot tell how many neighbours have broadcast, or which ones.

Our algorithm is remarkably simple and computes a greedy colouring in expected $O(\Delta + \log n)$ time, where n is the number of nodes and Δ is the maximum degree of the network. Once again the expected total number of messages sent by each processor is bounded by a constant. As well as matching the best known worst-case time complexity for obtaining a greedy colouring, our algorithm is the first proposed algorithm for this problem where the nodes require no prior knowledge of the network and do not need to distinguish between their neighbours.

To obtain our results we introduce a new form of analysis to determine the time complexity. Nearly all previous analytical techniques for distributed MIS selection algorithms have relied on a general technique, which divides the computation into successive phases and shows that some fixed fraction of the network is expected to be eliminated in each phase, so that there are at most logarithmically many phases [Lub86, Lyn96]. Our algorithms do not have this property, and hence require a more flexible form of analysis, which we describe in detail in Chapter 6. The related results of Chapter 6 are summarized in [JSX14] which has been submitted for publication.

In Chapter 7, we give some experimental results on different types of graphs to illustrate the time complexity and message complexity of our proposed algorithms in practice. In addition, we experimentally test and compare the size of the MIS chosen by our MIS selection algorithms on some standard maximum independent set benchmarks. We also experimentally test and compare the number of colours used by our GC algorithms on some standard graph colouring benchmarks. We show experimentally that, the MIS algorithms do not perform very well on maximizing the size of MIS chosen (they are not designed to do so), however, the number of colours used by our distributed greedy colouring algorithms turns out to be optimal or near-optimal for many standard graph colouring benchmarks, so they provide effective heuristic approaches to computing a colouring with a small number of colours.

Finally, in Chapter 8, we summarize the main contributions of this thesis and draw some conclusions from our results as well as identifying some open questions and future research directions.

CHAPTER 2

Background

In this chapter, we give the necessary background on distributed and parallel computing, cellular computing and some classical computational problems to be tackled by cellular computing in this thesis. Cellular computing aims at getting inspiration from biological cells to improve the computational performance on solving distributed and parallel computing problems. This chapter focuses on the background of cellular distributed and parallel computing: In section 2.1, we introduce the background of traditional approaches to distributed and parallel computing. In Section 2.2, we first introduce a theoretical cellular computing model – P systems, then we introduce the concept of cellular computing and a restricted general cellular computing model. In Section 2.3, we introduce and review some classical computational problems that are studied in the thesis: the constraint satisfaction problem, the distributed maximal independent set selection problem and the distributed greedy colouring problem.

2.1 Distributed and parallel computing

In distributed computing, networked computers are required to communicate with their neighbours in order to achieve a common goal across the whole network; In parallel computing, parallel computations are carried out simultaneously by several collaborative computers based on the principle that large problems can often be divided into smaller ones that are then solved concurrently.

It is possible to roughly classify concurrent systems as "distributed" or "parallel" using the following criteria [Pap03]:

- In distributed computing, each computer has its own private distributed memory and information is usually exchanged between them by exchanging messages with their neighbouring

computers.

- In parallel computing, all computers may have access to a shared memory to exchange information between computers.

Nevertheless, distributed computing and parallel computing have a lot of overlap: a computing system may be characterised as both distributed and parallel. For example, in most distributed network deployment, distributed computers (processors) exchange messages and work in parallel to achieve a common goal. So distributed computing and parallel computing are sometimes mentioned together as *distributed and parallel computing*.

The study of distributed and parallel computing has a long history [BJL$^+$82, AFL83, Upf84, FLP85] and it's regarded as an important branch of computer science [Cri88]. Application examples of distributed and parallel computing vary from wireless sensor networks [LW12] to online social networks [MMG$^+$07] and traffic flow management [TA07].

There are several classical computation models of distributed and parallel computing, such as the model of parallel random access machines (PRAM) where distributed processors can get access to a shared memory [CLRS09] and the much more recent model of networked finite state machines (nFSM) which depicts a network of finite state automata [EW13].

One of the most widely-used models of distributed and parallel computing is a message passing model known as the Linial model [Cha87, Lin86, Lin92, GKK$^+$09]. In Linial's model, a distributed network consists of a set V of processors and a set E of bidirectional communication links (channels) between pairs of processors. Two processors are neighbours if there is a communication link connecting them. A distributed network with n processors where each processor has at most Δ neighbouring processors corresponds to an undirected graph $G = (V, E)$ with n vertices and maximum degree Δ. A network is called *anonymous* if the processors cannot distinguish each other by unique identifiers. Linial's model of distributed computing is a synchronous system where all vertices operate in a lockstep fashion. During each round of communication, all vertices act in parallel and carry out the following steps sequentially [Lyn96]:

1. receive the latest messages from their neighbours;
2. perform arbitrary local computation;
3. send new messages to their neighbours.

The computation is said to be complete only when the local computations at every vertex have halted.

In such systems, the main complexity measure for both deterministic and randomised algorithms is the number of synchronous communication rounds required to complete the computation.

Another complexity measure that is also considered in some situations is the number of messages communicated, or their total size (in bits) [Lyn96].

2.2 P systems and cellular computing

The concept of *bio-inspired computing* can be dated back more than half a century to the original musings on artificial intelligence by Alan Turing as well as the early work on self-replicating cellular automata by John von Neumann in the 1940s [Kep11]. There has been considerable fruitful research in the area of bio-inspired computing which has had great impact on computer science since then. Some successful examples of bio-inspired computing include *cellular automata* [Wol94], *artificial neurons* [MP43], *neural networks* [Bis95], *evolutionary computation* [BFM97] and *swarm intelligence* [Eng05].

Cells are the basic information processing units of life. As such, computer scientists have frequently turned to biological cells for inspiration [PRS10, AAB+11, Sip99, LK99, NZS+06]. Indeed, biological cells and computers share many similarities: they are both information systems which have memories, evolve over time and can be networked for communication. However, biological cells seem to have great power to tackle complicated tasks by using only simple operations. This is a very attractive feature that computer scientists have been seeking to replicate. This power of biological cells is particularly relevant in the area of distributed and parallel computing since biological cells are inherently distributed and parallel.

2.2.1 P systems

Membrane systems (also known as P systems) have been introduced as a class of parallel, nondeterministic, synchronous and distributed models of computation inspired by the structure and functioning of living cells [PRS10]. The basic model of P systems consists of a hierarchical structure composed of several membranes, embedded into a main membrane called the *skin*. Membranes divide Euclidean space into *regions*, that contain some *objects* (represented by symbols of an alphabet) and evolution *rules*. Using these rules, the objects may evolve and/or move from a region to a neighboring one.

P systems have been widely used as a framework for *modeling* in many different areas including systems and synthetic biology, economics, linguistics, computer science, and optimization [CPPJ06, FGPJ14, RCPJ08, CdAPH+10, CCM+11, Xu12]. More information on P systems is available from the P systems webpage: `http://ppage.psystems.eu` and a comprehensive overview can be found in [PRS10].

One fundamental research topic is the *computational completeness/universality* of P systems [IMVPP03, PJ09, PJ10, PPJ10, PZZ11, PWH12]. There has also been considerable research into

the *computational efficiency* of P systems, and the possible trade-offs between different resources. Alhazov et al. have shown that P systems with membrane division can be used to solve **PSPACE**-complete problems in linear time by rapidly generating an exponential work space [AMVP03]. Ishdorj et al. have shown that spiking neural P systems with an exponential amount of pre-computed resources can be used to solve **PSPACE**-complete problems in polynomial time [ILP$^+$10]. Pan et al. have shown how spiking neural P systems with neuron division and budding can be used to solve **NP**-complete problems in polynomial time [PPPJ11]. Many other papers on the computational efficiency of different variants of P systems can also be found in the literature [LZFM07, DPGNPJRNn08, XLZ10, XZJ13].

With respect to the computational efficiency of P systems, a large variety of P systems have been proposed to solve **NP**-hard problems in polynomial time. These solutions are usually obtained by generating an exponential amount of workspace in polynomial time and using parallelism to check simultaneously all the candidate solutions. For cell-like P systems, three kinds of operations on membranes have been proposed to obtain exponential workspace in polynomial time: membrane division (mitosis) [PĎ00], membrane creation (autopoiesis) [IMVP01] and membrane separation (membrane fission) [PI04]. Each of these three operations is inspired by the related biological operation. The corresponding P system models (P systems with membrane division, creation, and separation) have been successfully used to design systems that can be used to solve **NP**-hard problems in polynomial time (e.g., [XLZ10, GNPJRC05, PI04]).

In addition to cell-like P systems which consist of hierarchical arrangements of membranes, tissue membrane systems whose structures are described by graphs have also been investigated for computational efficiency [MVPPRP03]. In a tissue P system each cell is represented by one node of a graph and each interface between two cells is represented by an edge of the graph. If an interface between two cells exists, then they can communicate with each other according to rules associated with each of the cells. Two broad classes of tissue P systems have been proposed which generate exponential workspace in polynomial time to solve **NP**-hard problems: tissue P systems with cell division [PPJRN08] and tissue P systems with cell separation [PPJ10].

Recently P systems have been used as distributed and parallel computing models to implement some classic distributed and parallel algorithms [Nic12]. For example, Nicolescu et al. describe a distributed algorithm for determining a maximum cardinality set of edge- and node-disjoint paths between a source cell and a target cell that is implemented by P systems [NW11]. The solution leverages the distributed and parallel characteristics of P systems. This work is similar in spirit to our work, but focuses on a different computational problem from the computational problems we investigate in this thesis. Similar to the neural-like P systems defined in Definition 3.2.1, the P systems defined by Nicolescu et al. in [NW11] also rely on the use of states for each cell and work in the max mode of rule application and in the $repl$ manner of communication.

2.2.2 Cellular computing

Definition 2.2.1 (Cellular Computing)**.** *Cellular computing is short for cellular distributed and parallel computing which represents new computational approaches to distributed and parallel computing that are inspired by the mechanisms and functioning of biological cells.*

At its heart, cellular computing consists of three principles: *simplicity, parallelism,* and *locality* [Sip99]. These three principles are illustrated in Figure 2.1. Cellular computing aims at solving problems by using a group of simple processors working altogether in a distributed and parallel way. In cellular computing, each processor is so simple that it can often be regarded as a finite-state machine with only local information about the computation, nevertheless, all the processors together perform high-efficiency computation to solve hard computational problems.

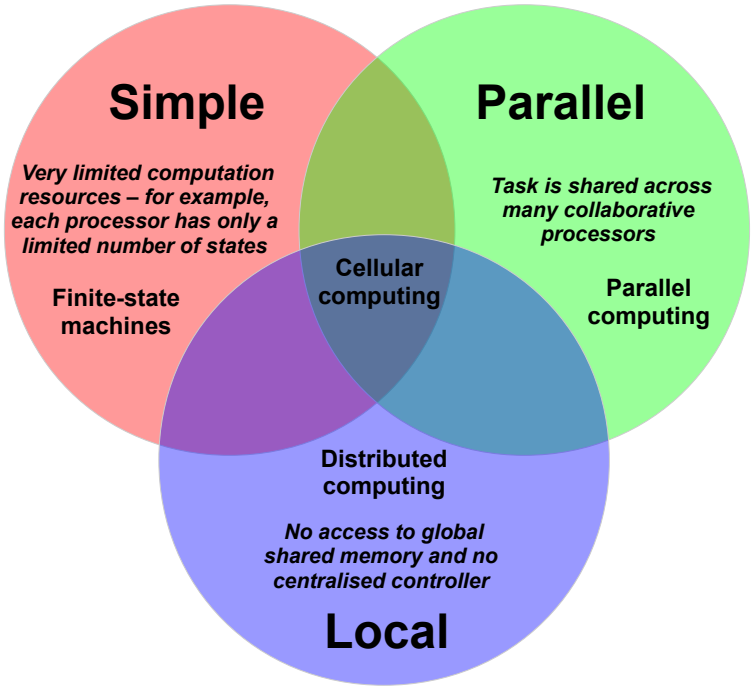


Figure 2.1: Three principles of cellular computing

It is clear that biological systems are generally asynchronous, that is, each cell operates independently with no global clock. For example, the biological system controlling the selection of sensory organ precursor cells in the developing fly brain, studied in [AAB$^+$11], is asynchronous in this sense. However, as stated in [AAB$^+$11]:

> "Certain aspects of the biological system resemble synchronous models and are not generally assumed in asynchronous computational systems. These synchronous like characteristics include: 1. coordinated start time (wake-up): cells start the SOP selection process based on an external biochemical signal that reaches all cells at roughly the same time. This is hard to achieve in a completely asynchronous computational system. 2. messages have a known delay: This is the major difference between synchronous and asynchronous computational models (the latter models do not assume a bound on message delay). In the biological system the messages involve a physical interaction using a biochemical process (Notch-Delta signaling). This process has a predictable delay which can be used to tune the internal selection probabilities."

P systems can be regarded as a theoretical model of cellular computing. Based on the Linial model, we follow [AAB$^+$11] in assuming a synchronous model with a global clock counting the number of steps of the computation and impose the following severe additional conditions to define a restricted general cellular computing model:

1. Each processor is anonymous and has minimal global information about the network;
2. At each time step, each processor either keeps silent or broadcasts one simple message to all its neighbours;
3. Each processor can tell whether at least one neighbour has broadcast a message, but cannot tell how many of them have done so, or which ones.

Information about a network may be difficult to obtain, or subject to uncertainty and change, so it is desirable for some applications to find algorithms that can complete their task without using such information [HM90, PRS97]. Moreover, using a small number of messages, each containing a single bit (or a small number of bits), allows an implementation to use less communication resources and less energy, and this may be crucial in some applications [LW12]. Because of the restrictions we impose, cellular computing approaches can be implemented using very simple communication mechanisms such as radio waves, optical signals, or even chemical signals, as in biological intercellular signalling [Bra06, CMML96].

By learning from biological cells, cellular computing promises to provide new approaches for conducting distributed and parallel computation more efficiently in terms of speed, cost, information storage and power dissipation, as well as solution quality. Simultaneously, given higher computing efficiency, cellular computing provides the possibility of addressing much larger problem instances than previously possible, at least for some application domains [Sip99, BHS09].

2.3 Some classical computational problems

In this section, we introduce some classical computational problems that are tackled by cellular computing in this thesis. These problems are the constraint satisfaction problem, the distributed maximal independent set selection problem, and the distributed greedy colouring problem.

2.3.1 Constraint satisfaction problems

The classic definition of the *constraint satisfaction problem* is as follows.

Definition 2.3.1 (Constraint Satisfaction Problem [RBW06])**.** *An instance of the Constraint Satisfaction Problem (CSP) is a triple $\mathcal{P} = (X, D, C)$ where*

- $X = \{x_1, x_2, \ldots, x_n\}$ *is a set of variables;*
- $D = \{D_1, D_2, \ldots, D_n\}$ *is a set of domains of possible values for those variables, such that the values assigned to x_i must belong to D_i;*
- $C = \{C_1, C_2, \ldots, C_t\}$ *is a set of* constraints.

Each constraint C_j is a pair (R_{S_j}, S_j) where S_j is a subset of the variables X, called the scope *of C_j, and R_{S_j} is a* relation *on the variables in S_j. In other words, each R_{S_j} is a set of tuples which is a subset of the Cartesian product of the domains of the variables in S_j.*[1]

A *solution* to a CSP instance is an assignment of values to all of the variables that satisfies all the constraints. Each value assigned to a variable must be a member of the corresponding domain for that variable. An assignment satisfies a constraint if the combination of values assigned to the variables in the scope of the constraint is allowed by the constraint relation.

In a given task one may be required to find the set of all solutions, $sol(\mathcal{P})$, to determine if that set is non-empty, or just to find any solution, if one exists. If the set of solutions is empty the CSP instance is unsatisfiable.

A large number of problems in artificial intelligence and other areas of computer science can be viewed as special cases of the Constraint Satisfaction Problem (CSP) [RBW06]. Examples include machine vision [Cha79], belief maintenance [Dec87], scheduling [DR90], temporal reasoning [All84], graph problems [LV02] and satisfiability problems [Zab88].

[1] This is the conventional definition, which we will adhere to here. A more parsimonious definition of the CSP would dispense with D entirely, leaving the role of each D_i to be played by a unary constraint C_j with $S_j = \{x_i\}$.

Example 2.3.2. *The graph coloring problem can be viewed as a special case of the CSP. In this problem, each node of a given graph needs to be coloured (using a given set of colors) such that no two adjacent nodes have the same colour [Kar72] (see Definition 2.3.6).*

To represent this problem as a CSP, each node in the graph is associated with a variable, and the domain of values for each of these variables is the given set of colours. There is a constraint on each pair of adjacent nodes (variables), imposing the restriction that they must be assigned different colours (values).

Generally, solving the CSP is **NP**-hard [RBW06]. Many specific CSP problems, such as SAT [XLZ10], the 3-Coloring Problem [WMSZ09], and the Hamiltonian Path Problem [MVPPRP03] have been tackled by different classes of P systems in the literature. However, to the best of our knowledge, no solution to the general CSP by P systems has previously been proposed.

2.3.2 Distributed maximal independent set selection

A well-established problem in distributed computing is that of electing a set of local leaders in a network of connected processors, known as the *maximal independent set* (MIS) selection problem. An MIS is a maximal set of nodes in a network such that no two of them are neighbours. Since the set is maximal, every node in the network is either in the MIS or a neighbour of a node in the MIS. Distributively selecting an MIS has been extensively studied in various models [Lub85, Lin92, ABI86, KMNW05, KMW06, MW05]. It has many applications in networking, and in particular in radio sensor networks. These include the construction of a backbone for wireless networks [Kro11], a foundation for routing and for clustering of nodes, and generating spanning trees to reduce communication costs [Pel00].

Definition 2.3.3 (Maximal Independent Set)**.** *Given an undirected graph $G = (V, E)$, an* independent set *in G is a subset of vertices $U \subseteq V$, such that no two vertices in U are adjacent. An independent set U is called a* maximal independent set *(MIS) if no further vertex can be added to U without violating independence.*

Example 2.3.4. *An example of a maximal independent set is shown in Figure 2.2 for a random undirected graph with 20 nodes. The set of vertices $v_8, v_{13}, v_{14}, v_{17}$ is a maximal independent set because no two nodes in this set are adjacent, and no further node of the graph can be added to this set without violating this property.*

Example 2.3.5. *Another example of a maximal independent set is shown in Figure 2.3 for a graph constructed from UK cities. The nodes of the graph correspond to 73 major cities in the UK, and two nodes are adjacent if the corresponding cities are less than 100km apart. Every major city in*

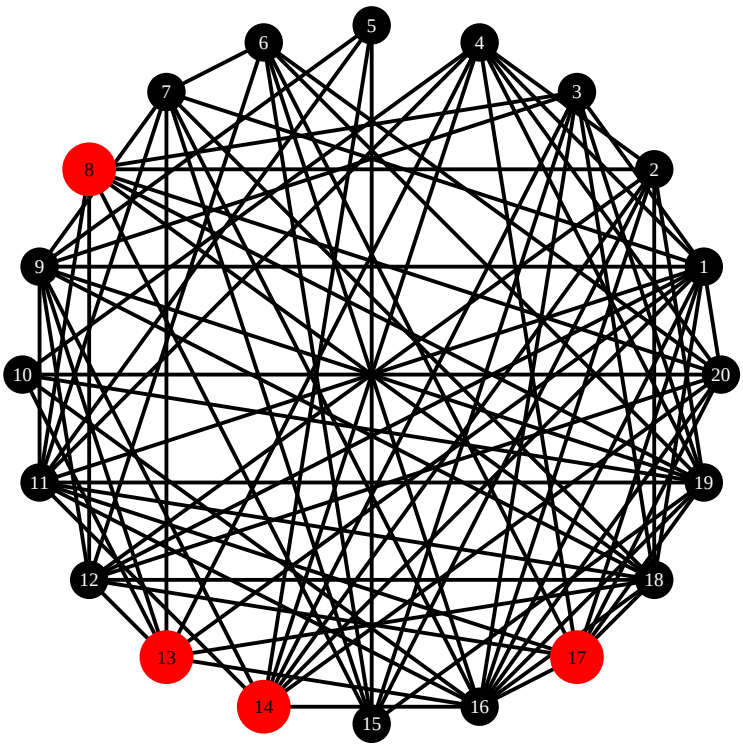


Figure 2.2: An example of a maximal independent set on a random graph

the UK is therefore within 100km of one of the 14 cities selected to be in the MIS (which are shown highlighted in the figure).

Different maximal independent sets for the same network can vary greatly in size. In contrast to the MIS selection problem, the related problem of finding a *maximum size* independent set (MaxIS) is notoriously hard. It is equivalent to finding a maximum clique in the complementary graph, and is therefore **NP**-hard [Kar72]. However, computing an arbitrary MIS (which is not necessarily of the maximum possible size) in linear time using a centralised sequential algorithm is trivial: simply scan the nodes in arbitrary order. If a node u does not violate independence, add u to the MIS. If u violates independence, discard it. Hence the challenge is to compute such an MIS more efficiently in a distributed way with no centralised control.

The study of distributed MIS selection can be traced back to the 1980s [AAB$^+$11]. It was shown early on that the MIS selection problem is in the complexity class **NC** [KW85], and hence likely to be a good candidate for a parallel or distributed approach.

We review the current state-of-the-art for distributed MIS selection here, focusing attention on the size of the messages (in bits) and the information about the network that is needed at each node (see Table 2.1).

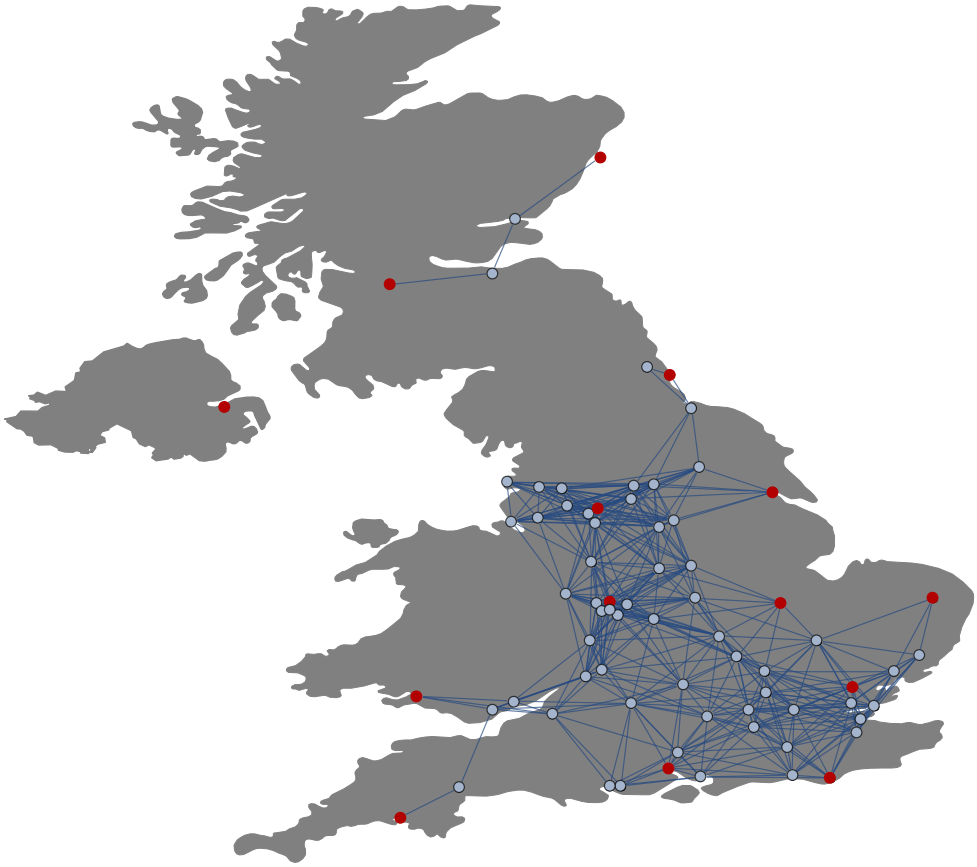


Figure 2.3: An example of an MIS selected on a network constructed from UK cities

A lower bound of $\Omega(\log^* n)$ time for distributed MIS selection on graphs with maximum degree $\Delta \geq 2$ is given in [Lin87]. The most well-known lower bound for distributed MIS selection on general graphs, $\Omega(\sqrt{\log n/(\log\log n)})$, is given in [KMW04]. Another known result shows that there is a graph with n vertices and with maximum degree $2^{\Theta(\sqrt{\log n})}$ where any (even randomised) distributed MIS selection algorithm requires $\Omega(\sqrt{\log n})$ time [Pel00]. This gives a lower bound of $\Omega(min\{\log\Delta, \sqrt{\log n}\})$ for distributed MIS selection on general graphs [BEPS12]. Most recently, in [MRSDZ11], it has been shown that if only one-bit messages are allowed to be sent along each edge in any time step, then every distributed algorithm to select an MIS in a ring of size n requires at least $\Omega(\log n)$ time steps with high probability.

For deterministic distributed MIS selection on general graphs, the fastest known algorithms run in $O(\Delta + \log^* n)$ time [Kuh09, BE09] or $O(2^{O(\sqrt{\log n})})$ time [PS96]. These deterministic MIS algorithms rely on very sophisticated multi-phase techniques, use a considerable amount of global information about the graph at each node, including unique node IDs, and allow complex messages to be sent on specific channels between nodes. Note that any deterministic algorithm requires some information at each node (such as a unique node ID) in order to break the symmetry [IR90, Pel00].

Table 2.1: Distributed MIS selection algorithms on general graphs with n nodes and maximal degree Δ

Type	Rounds/Time	Message size (bits)	Information required at each node	Reference
Det.	$O(\Delta + \log^* n)$	$\Omega(\log n)$	Unique ID, size and maximum degree of the graph, and channel names	[BE09]
				[Kuh09]
	$O(2^{O(\sqrt{\log n})})$			[PS96]
Rand.	$O(\log^2 n)$	$\Omega(\log \Delta)$	Maximum degree in 2-neighbourhood	[Pel00]
		3	None	[EW13]
		1	Size of the graph	[AAB$^+$11]
		1	None	[AABJ$^+$11]
	$O(\log \Delta \sqrt{\log n})$	$\Omega(\log n)$	Size, maximum degree of the graph, and channel names	[BEPS12]
	$O(\log n)$	$\Omega(\log n)$	Size of the graph	[Lyn96]
		$\Omega(\log \Delta)$	Degrees of neighbours	[Wat07]
		1	Channel names	[MRSDZ11]
		1	**None**	**Table 6.1**

Using randomisation to break the symmetry between nodes allows for simpler algorithms, often requiring a smaller number of time steps. A simple parallel randomised algorithm for distributed MIS selection in the PRAM model of computation was presented in 1986 by Luby [Lub86] and independently by Alon et al. [ABI86].

This algorithm has been adapted to the message-passing model of distributed computation in several slightly different ways. In the version presented by Lynch [Lyn96] each processor is assumed to know the total size, n, of the graph, and chooses a random integer in the range 1 to n^4 at each time step. These integers are then broadcast as messages to all neighbouring nodes, so the messages sent between processors contain $\Omega(\log n)$ bits. Using these messages the nodes are able to compute an MIS by selecting the nodes that choose the largest random values in their neighbourhood, removing those nodes and their neighbours, and iterating this process. Using the analysis from [Lub86], this process is shown to terminate in $O(\log n)$ time on average and with high probability.

In the version presented in [Wat07] the nodes choose a probability value based on their degree in the graph, and use this value, together with the degree values of their neighbours to decide whether to join the MIS at each time step. In this variant the nodes must exchange messages to determine the current degrees of their neighbours at each time step, and hence the messages sent between processors contain $\Omega(\log \Delta)$ bits. Once again, using the analysis from [Lub86], this process is shown to terminate in $O(\log n)$ time on average and with high probability.

In the version presented in [Pel00] each vertex chooses a probability value based on the max-

imum degree of the nodes at distance 1 or 2 away from it in the graph, and then uses this value to decide whether to join the MIS at each time step. Peleg shows with a simpler analysis that this algorithm halts in $O(\log^2 n)$ time on average and with high probability. Once again the nodes must exchange messages to determine the current degrees of their neighbours at each time step, and hence the messages sent between processors contain $\Omega(\log \Delta)$ bits.

These distributed randomised algorithms, all based on a similar approach and generally known as Luby's algorithm, remained the state-of-the-art for many years, but there have recently been some new developments.

A new randomised MIS algorithm requiring $O(\log \Delta \sqrt{\log n})$ time was proposed in [BEPS12]. This algorithm improves on the $O(\log n)$ algorithms when $\log \Delta < \sqrt{\log n}$. However, this algorithm requires each processor to know the size and maximal degree of the graph and assumes that each processor can distinguish between channels so that it can send different messages along different edges. Since it relies on exchanging information about specific nodes, using node identities, the messages exchanged in this algorithm contain $\Omega(\log n)$ bits.

Algorithms for MIS selection on special graphs such as sparse graphs and growth-bounded graphs have also been studied [GPS88, BE10, SW08].

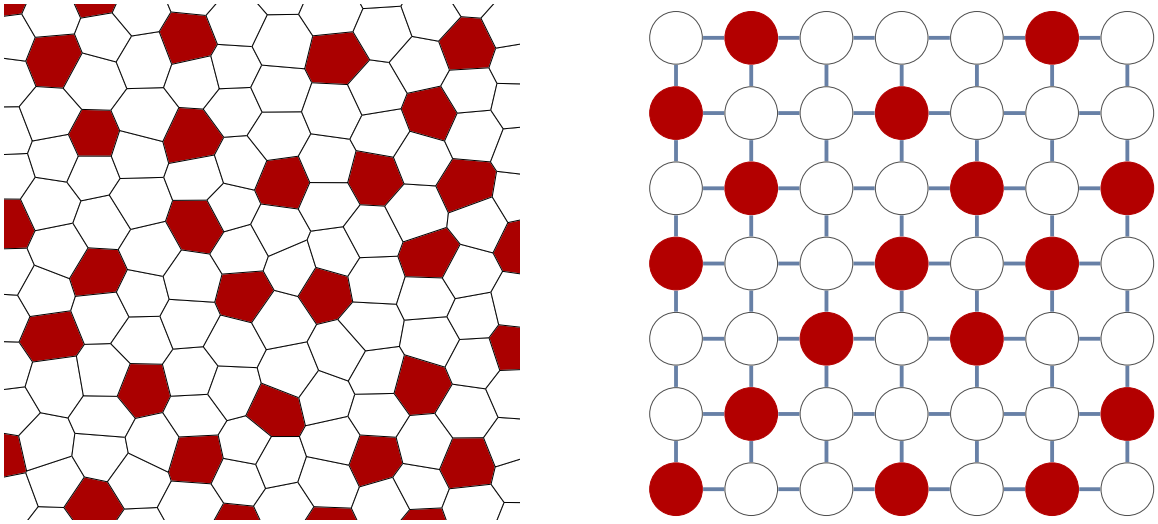
There has been a recent spate of interest in finding efficient distributed MIS selection algorithms that can work in more restricted computational models, such as wireless network models [MW05, EW13, CK10, AAB$^+$11, AABJ$^+$11, MRSDZ11, XJ13].

For example, the approach proposed in [MRSDZ11] splits the randomly generated values at each node into single bits, and communicates them one by one. When these bits are broadcast to all neighbours, this approach achieves a time complexity of $O(\log^2 n)$. By distinguishing between different neighbours, and having separate, overlapping, exchanges of messages with each neighbour, the overall time complexity is brought down to $O(\log n)$ time on average and with high probability. This is shown to be the optimal time complexity that can be achieved with one-bit messages [MRSDZ11]. However, to achieve this optimal performance requires that each vertex can distinguish between its neighbours by locally known channel names, so that different messages can be sent along different edges at the same time step.

A more radical approach is the novel distributed MIS selection algorithm inspired by the neurological development of the fruit fly which is given in [AAB$^+$11, AABJ$^+$11].

During development, certain cells in the pre-neural clusters of the fruit fly specialise to become sensory organ precursor (SOP) cells, which later develop into cells attached to small bristles (microchaetes) on the fly that are used to sense the environment. During the first stage of this developmental process each cell either becomes an SOP or a neighbour of an SOP, and no two SOPs are neighbours. These observed conditions are identical to the formal requirements in the maximal independent set selection problem (see Definition 2.3.3).

The similarity of SOP cell selection and MIS selection in a graph is illustrated in Figure 2.4.



(a) The selection of SOP cells in the fruit fly

(b) An MIS selected from a 7×7 grid graph

Figure 2.4: The similarity between SOP cell selection and MIS selection

However, Afek et al. pointed out that the method used by the fly to select the SOPs appears to be rather different from the standard algorithms for choosing an MIS described above. The cells of the fly appear to solve the problem using only simple local interactions between certain membrane-bound proteins, notably the proteins Notch and Delta [Bra06, CMML96]. Moreover, they require very little knowledge about connectivity.

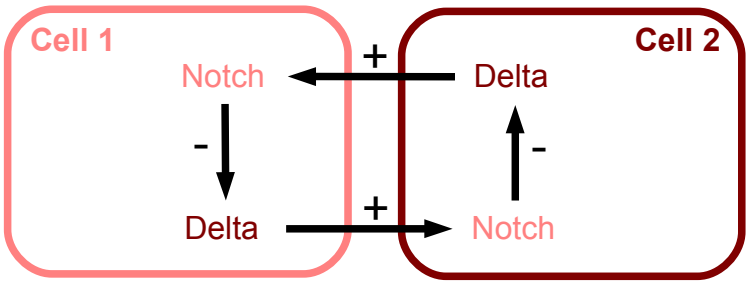


Figure 2.5: A positive feedback constructed by Notch-Delta signalling

The Notch-Delta signalling pathway provides a communication channel between neighbouring cells during development. It is thought to play a critical role in the formation of "fine-grained" patterns in the development of many organisms [SLL$^+$10, GKAT12], helping to generate distinct cell fates among groups of initially equivalent neighbouring cells. In particular, many studies have shown that Notch-Delta signalling regulates the selection of *Drosophila* neural precursors from groups of equipotent proneural cells in a way which resembles the MIS selection problem [BRHB10, BHB11,

SLL$^+$10]. The transmembrane protein Delta has been shown to have two activities: Delta in one cell can bind to, and transactivate, the transmembrane protein Notch in its neighbouring cells [BRHB10]; Delta and Notch in the same cell mutually inactivate each other [JBAM98, MC09]. The interaction of the transmembrane proteins Notch and Delta therefore constitutes a positive feedback mechanism which generates an ultrasensitive switch between two mutually exclusive signalling states: *sending* (high Delta/low Notch) and *receiving* (high Notch/low Delta). A slight excess of Delta production in one cell can generate a strong signalling bias in one direction: the cell becomes a sender and its neighbours become receivers [SLL$^+$10]. At the multicellular level, this lateral inhibition mechanism can break the symmetry among cells and amplify small differences between neighbouring cells, thus facilitating pattern formation (see Figure 2.5).

As shown in Figure 2.5, Notch-Delta signalling can set up a positive feedback to amplify small differences between cells. A slight excess of Delta in cell 2 enables the positive feedback loop that leads to mutually exclusive signalling states of the two cells.

Based on their study of this developmental process, Afek et al. proposed an algorithm that works in a distributed model where each node can only broadcast to all its neighbours or remain silent. Moreover, each node can only detect whether at least one neighbour has broadcast a signal. This model of communication is sometimes referred to as a "beeping" model with collision detection [AABJ$^+$11].

The algorithm proposed by Afek et al. is fully synchronous and operates over discrete time steps. At each step, each node may choose, with a certain probability (that varies over time), to signal to all its neighbours that it wishes to join the independent set. If a node chooses to issue this signal, and none of its neighbours choose to do so in the same time step, then it successfully joins the independent set, and becomes inactive, along with all its immediate neighbours. However if any of these neighbouring nodes issue the same signal at the same time step, then the cell does not succeed in joining the independent set at that step. This process is repeated until all nodes eventually become inactive. This algorithm is remarkably simple: it requires no knowledge about the number of active neighbours and uses only one-bit messages. The computation at each individual node can be described by a simple automaton, as shown in Figure 2.6: At each step a node may signal that it wishes to join the independent set by moving to the state at the top right with probability p (which varies with time). It then responds to the signals from neighbouring cells.

In their proposed algorithm, each node broadcasts at each time step with a certain probability, which changes over time, and then checks whether any of its neighbours has broadcast at the same time. As originally presented [AAB$^+$11], the algorithm uses a sequence of gradually increasing global probability values calculated from the total number of nodes of the graph n and its maximum degree Δ. The algorithm was further refined by Afek et al. in a later paper [AABJ$^+$11]. In the later version the probability values are chosen according to a fixed pattern, so that the individual nodes

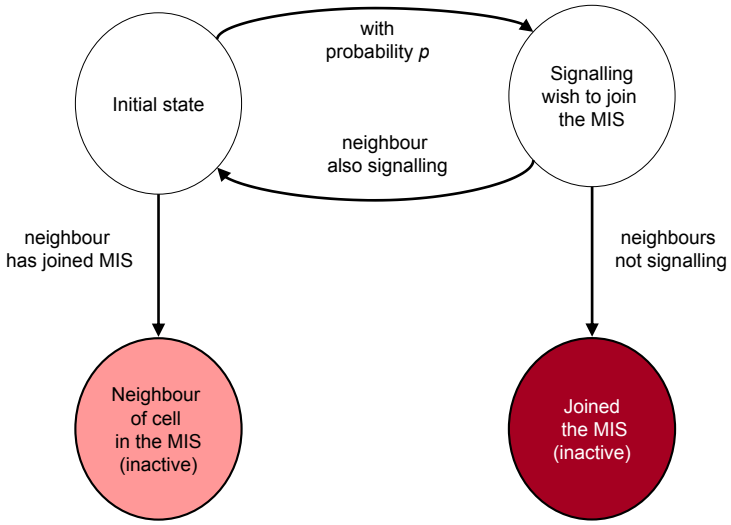


Figure 2.6: Automaton description of the pattern formation process at each cell

require no information at all about the graph. However, in both versions the expected number of time steps required was shown to be $O(\log^2 n)$ (see Table 2.1).

Another recent approach to distributed computing with very restricted communication and processing capabilities is the networked finite state machine model introduced in [EW13]. This only allows a fixed finite number of distinct messages, and very limited computation at each node, based on the notion of a randomised finite state machine, with no information about the network. It is shown in [EW13] that an MIS can be computed in this very restricted model in $O(\log^2 n)$ time, using only 7 states and 7 corresponding messages.

2.3.3 Distributed greedy colouring

One of the most fundamental graph-algorithmic problems is to colour the vertices of a graph so that adjacent vertices are assigned different colours. This problem is known as the *graph colouring* problem [Kar72].

A *proper colouring* of a graph is a function that assigns a colour to each vertex such that no two adjacent vertices are assigned the same colour. The colouring is called a k-colouring if at most k different colours are used.

Definition 2.3.6 (Graph Colouring)**.** *Given an undirected graph $G = (V, E)$, a* k-colouring *of G is a function $f : V \longrightarrow \{c_1, c_2, \ldots, c_k\}$ such that $f(u) \neq f(v)$ for every edge $\{u, v\} \in E$. The elements of the set $\{c_1, c_2, \ldots, c_k\}$ are called* colours*. G is called* k-colourable *if and only if there exists a k-colouring of G.*

For many practical applications it is desirable to minimize the number of colours used. The smallest possible positive integer k for which there exists a k-colouring of G is defined to be the *chromatic number* of G, usually denoted by χ. It is known to be **NP**-hard to approximate the chromatic number χ within a factor of $|V|^{1-\varepsilon}$, for any $\varepsilon > 0$, even using a centralised algorithm with complete knowledge of the graph [Zuc06].

However, a number of heuristic approaches can be used to rapidly obtain colourings with a reasonably low number of colours on many graphs. For example, the following greedy approach produces a colouring in linear time using a centralised control.

Definition 2.3.7 (Greedy Colouring)**.** *Given an arbitrary ordering, $(v_1, v_2, \ldots, v_n)$, of the vertices of G, and an arbitrary ordering on the colour values, a* greedy colouring *algorithm considers each vertex from v_1 to v_n in turn, assigning each vertex the smallest possible colour value that is not already assigned to any of its neighbours.*

Note that a colouring obtained in this way has the property that no individual vertex can be recoloured using a smaller colour. A colouring with this property is sometimes called a *Grundy colouring* [GKK$^+$09, Gru39]. Since every greedy colouring algorithm produces a Grundy colouring, and every Grundy colouring can be obtained using a greedy colouring algorithm (by choosing a suitable ordering on the vertices) [GKK$^+$09], we will refer to any Grundy colouring as a greedy colouring, even if it is computed in some other way.

It is easy to see that a greedy colouring uses no more than $\Delta + 1$ colours, so we have that $\chi \leq \Delta + 1$ for any graph G.

Example 2.3.8. *Consider the network shown in Figure 2.7. As in Example 2.3.5, this network is constructed from UK cities: the vertices of the network correspond to 73 major cities in the UK, and two vertices are adjacent if the corresponding cities are less than 100km apart. The graph of this network has maximum degree 25, and can be coloured with 13 distinct colours as shown in Figure 2.7.*

The assignment of colours shown in Figure 2.7 has the property that no two neighbouring vertices share the same colour. Moreover, if we assume that the colours are ordered as shown in the colour bar at the top right of the figure, each vertex with a colour that is not first in the ordering is adjacent to at least one vertex with each lower-ordered colour. Hence no vertex can be assigned a colour that is lower in the ordering without violating the property that neighbouring vertices are assigned different colours. It follows that the colouring shown in Figure 2.7 is a greedy colouring.

There are many good reasons to seek greedy colourings from both theoretical and practical viewpoints. Any graph G has at least one greedy colouring which uses the optimal number, χ, of colours [HJS02]. For some special types of graphs, any greedy colouring will use an optimal

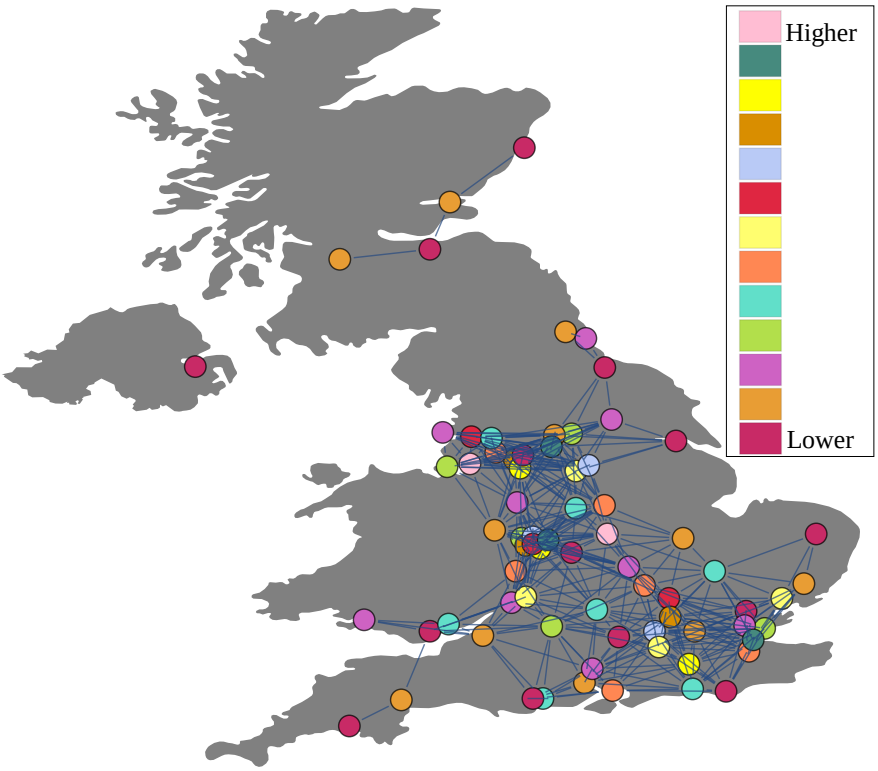


Figure 2.7: An example of a greedy colouring on a network constructed from UK cities

or near-optimal number of colours. For example, any greedy colouring of a k-partite complete graph will use k colours, which is the smallest possible number for this form of graph. For random graphs where each pair of vertices is joined by an edge with probability $\frac{1}{2}$, it has been shown that a greedy colouring will in expectation use only around twice as many colours as are necessary [GM75, HJS03].

To obtain a greedy colouring efficiently without any central control, using only local processing at each vertex and messages along the edges, is a challenging task, and is an important problem in distributed computing [GKK$^+$09]. Distributed greedy colouring serves as a basic building block in many other distributed algorithms, and has applications in networking, wireless sensor networks and in particular in frequency assignment in radio-communication networks [MP12, PL96, Wat05, NSW11].

The problem of $(\Delta + 1)$-colouring is closely related to MIS selection [KW06]. Hence it can be shown that the lower bounds for distributed MIS selection mentioned above also apply for distributed $(\Delta+1)$-colouring. Similarly, the state-of-the-art distributed $(\Delta+1)$-colouring algorithms are mostly obtained from the best algorithms for distributed MIS selection (see Table 2.2).

A randomised distributed $(\Delta + 1)$-colouring algorithm requiring $O(\log \Delta + \sqrt{\log n})$ time is proposed in [SW10]. In this algorithm the messages represent randomised preference levels for

each of the possible colours. This algorithm needs to know an upper bound of the size of the graph and requires each processor to be able to send different messages along different channels. Since the messages exchanged represent colours, the message size is $\Omega(\log \Delta)$ bits.

An algorithm with expected $O(\log \Delta + 2^{O(\sqrt{\log \log n})})$ time is given in [BEPS12]. However, this algorithm relies on a deterministic algorithm to complete a partial colouring, and hence requires unique node IDs, and messages with $\Omega(\log n)$ bits.

Johansson proposed and analysed a simple randomised distributed $(\Delta + 1)$-colouring algorithm requiring $O(\log n)$ time [OJ99]. The algorithm of Johansson requires that each vertex knows the maximum degree of the graph. Each message corresponds to a potential colour choice, so the messages in this algorithm contain $\Omega(\log \Delta)$ bits.

These algorithms do not attempt to obtain a *greedy* colouring, and hence tend to use the maximum number, $\Delta + 1$, of colours. For many classes of graphs, a greedy colouring will often use considerably fewer colours, but computing a greedy colouring with a distributed algorithm is a more challenging problem. In fact, the problem of computing a greedy colouring for a given ordering of the vertices is known to be **P**-complete [GHR95, GLGT03].

Panconesi and Rizzi proposed a deterministic algorithm for graph colouring that attempts to use a small number of colours [PR01]. This algorithm was not originally designed to construct a greedy colouring. However, it can be easily modified to become a distributed greedy colouring algorithm by always choosing the first available colour when assigning a colour. In view of this it is described in [GKK$^+$09] as the first distributed approach to greedy colouring. The number of time steps taken by this algorithm is $O(\Delta^2 + \log^* n)$ [PR01]. It is quite a sophisticated algorithm which relies on a preprocessing phase to produce a forest decomposition of the graph. It assumes that each vertex has a unique identifier, and it also requires that each vertex knows its own degree and the maximum degree of the whole graph. This algorithm also requires the ability to send different messages to different neighbours simultaneously. Because identifiers are exchanged the message size of the algorithm is $\Omega(\log n)$.

Hansen et al. proposed a randomised distributed algorithm for graph colouring in [HKKN04]. Even though the algorithm is not explicitly described in the original paper as a greedy colouring algorithm, it is pointed out in [GKK$^+$09] that the colourings it produces are actually greedy colourings. The expected number of time steps taken by this algorithm to produce a colouring is $O(\Delta^2 \log n)$ [HKKN04]. However, this algorithm assumes that each vertex knows its degree in the graph, and the messages exchanged include these numerical degree values as well as the colour values. Hence the size of each message sent is $\Omega(\log \Delta)$ bits.

Gavoille et al. give a detailed theoretical study of distributed greedy colouring [GKK$^+$09]. They establish a lower bound for this problem of $\Omega(\log n / \log \log n)$ time steps. Moreover, they show that an arbitrary k-colouring can be converted to a greedy colouring by a simple distributed algorithm

Table 2.2: Distributed $(\Delta + 1)$-colouring algorithms on general graphs with n nodes and maximal degree Δ

Greediness	Type	Rounds/Time	Message size (bits)	Information required at each node	Reference
Non-greedy	Det.	$O(\Delta + \log^* n)$	$\Omega(\log n)$	Unique ID, size and maximum degree of the graph, and channel names	[BE09]
					[Kuh09]
		$O(2^{O(\sqrt{\log n})})$			[PS96]
	Rand.	$O(\log \Delta + \sqrt{\log n})$	$\Omega(\log \Delta)$	Upper bound of the size of the graph and channel names	[SW10]
		$O(\log \Delta + 2^{O(\sqrt{\log \log n})})$	$\Omega(\log n)$	Unique ID, size and maximum degree of the graph, and channel names	[BEPS12]
		$O(\log n)$	$\Omega(\log n)$	Size of the graph	[Lyn96]
			$\Omega(\log \Delta)$	Degrees of neighbours	[Wat07]
				Maximum degree of the graph	[OJ99]
Greedy	Det.	$O(\Delta^2 + \log^* n)$	$\Omega(\log n)$	Unique ID, maximum degree of the graph, own degree and channel names	[PR01]
	Rand.	$O(\Delta^2 \log^2 n)$	$\Omega(\log \chi)$	**None**	**Table 5.2**
		$O(\Delta^2 \log n)$	$\Omega(\log \Delta)$	Own degree and degrees of neighbours	[HKKN04]
		$O(\Delta \log^2 n)$	$\Omega(\log \chi)$	**Size of the graph**	**Table 5.3**
		$O(\Delta + \log n)$	$\Omega(\log \Delta)$	Maximum degree of the graph	[GKK$^+$09]
			$\Omega(\log \chi)$	Channel names	[MRSDZ10]
				None	**Table 6.2**

in $O(k)$ time steps. However, this conversion algorithm requires each node to know the value of k. Combining this approach with the most efficient $(\Delta + 1)$-colouring algorithms described earlier gives a two-stage algorithm with an overall expected time complexity of $O(\Delta + \log n)$ when each node knows the value of Δ.

Métivier et al. proposed a simple randomised distributed $(\Delta + 1)$-colouring algorithm requiring $O(\Delta + \log n)$ time [MRSDZ10]. The algorithm proposed by Métivier et al., does not assume any global knowledge of the network, but requires each processor to know from which channel it receives each message. The algorithm consists of two stages: it first uses randomisation to break the symmetry and obtains a colouring in $O(\log n)$ time with an unbounded number of colours; it then reduces the number of colours used to at most $\Delta + 1$ in $O(\Delta + \log n)$ time. The colours in this second stage are chosen to be the smallest available, so the resulting colouring is a greedy colouring (although this is not made explicit in [MRSDZ10]). Each message exchanged in the first stage contains only one bit, but each message in the second stage represents a final colour choice. Since the

total number of colours used may be much lower than Δ in some classes of graphs, we give a lower limit on the message size of $\Omega(\log \chi)$ bits for this algorithm, where χ is the chromatic number.

CHAPTER 3

Neural-like P systems solving the CSP

In this chapter, we give a theoretical polynomial-time solution to the constraint satisfaction problem based on a specific class of P systems – neural-like P systems. This is the first proposed use of P systems to solve the general CSP. Our neural-like P systems work with strings: they broadcast symbol-impulses to neighbour cells using the *repl* manner of communication and process strings of symbol-impulses from neighbour cells using the *max* mode of rule application (see below). Our neural-like P systems can in principle solve an arbitrary CSP instance in polynomial time without using cell division or cell separation to create exponential workspace as in the conventional methods of using P systems to solve **NP**-hard problems.

We first recall some preliminaries needed in this chapter and the next. In the following two sections, the definition of neural-like P systems and recognizer neural-like P systems are described respectively. Then, a polynomial-time solution to the CSP by neural-like P systems is presented, including a short overview of the computation. Finally, a short summary is given.

3.1 Preliminaries

An alphabet, Σ, is a non-empty set, whose elements are called symbols. An ordered sequence of symbols is called a string. The number of symbols in a string u is called the length of the string, and is denoted by $|u|$. As usual, the empty string (with length 0) will be denoted by λ. The set of strings of length n built with symbols from the alphabet Σ is denoted by Σ^n and $\Sigma^* = \cup_{n\geq 0}\Sigma^n$.

A *multiset* m over a set A is a pair (A, f) where $f : A \to \mathbb{N}$ is a mapping. The values of the function f represent the number of times that each element of A is contained in the multiset. If $m = (A, f)$ is a multiset, then its *support* is defined as $supp(m) = \{x \in A | f(x) > 0\}$ and its *size* is

defined as $\sum_{x \in A} f(x)$. A multiset is empty (resp. finite) if its support is the empty set (resp. finite). A multiset, m_1 is said to be *contained in* a multiset m_2 if $m_1(x) \leq m_2(x)$ for all $x \in supp(m_1)$.

If $m = (A, f)$ is a finite multiset over A, then it will usually be convenient to represent m with a string over the alphabet A where each element a_i of $supp(m)$ appears $f(a_i)$ times. In this way, each string in A^* can be identified with a multiset over A.

3.2 Neural-like P systems

In this section we introduce the basic definitions of neural-like P systems. For more details about P systems and their variants see [MVPPRP03, IPY06, DPGOGN$^+$09]. Neural-like P systems are networks of cells inspired by the way biological neurons work together in order to process impulses in a complex network where they are linked through synapses [MVPPRP03, PRS10].

The class of neural-like P systems that we introduce here is a special class of tissue P systems where each cell has a state which controls the evolution of objects; the rules are rewriting rules of the form $su \longrightarrow s'v$, where s, s' are states, and $u \longrightarrow v$ is a standard string rewriting rule, with target indications in v, which direct the movement of objects from one cell to another. This is different from conventional tissue P systems which use communication rules in the form of *symport/antiport* rules [PPJRN08]. It is also different from Spiking Neural P systems where only the "spike" (a single object type) can be sent to neighboring cells [IPY06, WHP11, CFI$^+$06]. When moving between cells, the objects in a neural-like P system are replicated, and then sent to all neighboring cells which have an interface with the cell where the rule is applied. One of the cells in a neural-like P system is designated to be the output cell of the system that sends objects to the environment, providing an output for the computation.

The definition of a neural-like P system is as follows:

Definition 3.2.1 (Neural-like P System)**.** *A neural-like P system is a construct* $\Pi = (O, \sigma_1, \ldots, \sigma_n, syn, i_{out})$*, where:*

- O *is a finite alphabet, and its elements are called objects;*
- $syn \subseteq \{1, 2, \ldots, n\} \times \{1, 2, \ldots, n\}$ *are synapses connecting cells;*
- $i_{out} \in \{1, 2, \ldots, n\}$ *indicates the output cell;*
- $\sigma_1, \ldots, \sigma_n$ *are cells, of the form* $\sigma_i = (Q_i, s_{i,0}, w_{i,0}, R_i), 1 \leq i \leq n$*, where:*
  - Q_i *is a finite set of states for the cell* σ_i*;*
  - $s_{i,0} \in Q_i$ *is the initial state of the cell* σ_i*;*

- $w_{i,0} \in O^*$ *is the initial string of objects contained in the cell* σ_i*;*
- R_i *is a finite set of rules of the form* $sw \longrightarrow s'xy_{go}z_{out}$ *where* $s, s' \in Q_i, w, x \in O^*$*,* $y_{go} \in (O^* \times \{go\})$ *and* $z_{out} \in (O^* \times \{out\})$*, with the restriction that* $z_{out} = \lambda$ *(the empty string) for all* $i \in \{1, 2, \ldots, n\}$ *different from* i_{out}*.*

A neural-like P system operates as follows: For each cell σ_i the set of rules, R_i, contains rules of the form $sw \longrightarrow s'xy_{go}z_{out}$, where s, s' are states and w, x, y_{go}, z_{out} are strings over the alphabet of objects. A string is an ordered sequence of objects. When such a rule is applied to a string w in state s, this is replaced by x which stays in the current cell, y_{go} which goes to neighboring cells, and z_{out} which is sent out of the system. Specifically, we refer to the process of using rules of the form $sw \longrightarrow s'xy_{go}z_{out}$ where $x = wa$ or $y_{go} = (wa \times \{go\})$ or $z_{out} = (wa \times \{out\})$ as concatenating the symbol or string a to w. The string z_{out} is non-empty only for the output cell. Note that we allow some rules to have *priority* over other rules: When several rules can all be applied under one state, only those with the highest priority will be considered to be applied.

There are three modes of processing their objects for neural-like P systems introduced in [MVPPRP03] (denoted by *min* mode, *par* mode, *max* mode respectively). The *max* mode refers to the non-deterministic maximally parallel application of rules which requires that at each step, within the applicable rules with the highest priority in each cell, each object is assigned non-deterministically to only one such rule whose application relies on that object. All such rule assignments take place in a massively parallel way at each step, i.e., the rules chosen to be applied at each step must be applied. The rules involved in these transformations in each cell must all use the same states. From the biological point of view, the *max* mode is often considered the most realistic, as biological cells (neurons included) can process chemical objects (impulses) in a rather efficient – hence maximal – manner [MVPPRP03].

The communication rules refer to what happens to the objects occurring in the string y_{go} of a rule $sw \longrightarrow s'xy_{go}z_{out}$ when it is applied to a string w in a given cell. Three ways of communicating between cells for neural-like P systems are introduced in [MVPPRP03] (denoted the *repl* manner, *one* manner, and *spread* manner respectively). The *repl* manner, where each object occurring in y_{go} is sent to all the neighboring cells, has an intrinsic computational power since it can be used to generate an exponential number of objects in polynomial time [MVPPRP03].

Thus, in this chapter, we investigate the computational efficiency of neural-like P systems with the *max* processing mode and the *repl* communication manner. Readers can refer to [MVPPRP03, PRS10] for more details about the other processing modes and communication manners of neural-like P systems.

A *configuration* of a neural-like P system at step t, $\langle syn, (s_{1,t}, w_{1,t}), (s_{2,t}, w_{2,t}), \ldots, (s_{n,t}, w_{n,t})\rangle$, consists of a neural system structure (a set of cells with associated rules and the synapses connecting them), the states of all the cells and the contents of objects present inside all the cells. The initial configuration of a neural-like P system, $\langle syn, (s_{1,0}, w_{1,0}), (s_{2,0}, w_{2,0}), \ldots, (s_{n,0}, w_{n,0})\rangle$, consists of the neural system structure, the initial states of all the cells and the initial contents in all of the cells.

As usual for standard membrane systems, we assume the existence of a global clock which marks the timing of steps (single transitions) for the whole system.

A single transition from a configuration to a new one, i.e., from $\langle syn, (s_{1,t}, w_{1,t}), (s_{2,t}, w_{2,t}), \ldots, (s_{n,t}, w_{n,t})\rangle$ to $\langle syn, (s_{1,t+1}, w_{1,t+1}), (s_{2,t+1}, w_{2,t+1}), \ldots, (s_{n,t+1}, w_{n,t+1})\rangle$, is performed by applying the rules in each cell of the system in parallel. A cell remains unchanged during the transition if and only if no rules can be applied to it. A sequence of transitions, starting from the initial configuration $\langle syn, (s_{1,0}, w_{1,0}), (s_{2,0}, w_{2,0}), \ldots, (s_{n,0}, w_{n,0})\rangle$ to the configuration $\langle syn, (s_{1,t}, w_{1,t}), (s_{2,t}, w_{2,t}), \ldots, (s_{n,t}, w_{n,t})\rangle$ after t steps, is called a *computation* after t steps. A computation is said to be halting if it reaches a configuration where no rule can be applied anywhere in the system.

Normally, the result of a successful computation is the set of strings of objects emitted by the output cell i_{out} during the computation. Owing to the non-determinism in the choice of rules, one can get a set of possible (successful) computations, and thus a set of possible results.

We now present a simple example of a neural-like P system, in order to clarify the definitions and to illustrate the way in which these systems work.

Example 3.2.2. *Consider the neural-like P system* $\Pi = (O, \sigma_1, \sigma_2, \sigma_3, syn, i_{out})$*, where:*

- $O = \{a, b, r, e, s, u, l, t\}$*;*
- $syn = \{(1,2), (2,1), (1,3), (3,1)\}$*;*
- $\sigma_1 = (\{s, s'\}, s, a, \{sa \longrightarrow s'(a, go)(a, out), s'b \longrightarrow s'(result, out),\ s'a \longrightarrow s'(a, out)\})$*;*
- $\sigma_2 = (\{s\}, s, \lambda, \{sa \longrightarrow s(b, go)\})$*;*
- $\sigma_3 = (\{s\}, s, \lambda, \{sa \longrightarrow s(a, go)\})$*;*
- $i_{out} = \{1\}$ *indicates the output cell.*

This neural-like P system is graphically represented in Figure 3.1, where ovals represent each of the cells (these ovals contain the initial state, the initial objects, and the set of rules, and are labeled with $1, 2, \ldots, n$*), arrows indicate the synapses, and there is an arrow leaving from the output cell.*

We will now describe how this neural-like P system operates. Note we have an assumption that rule $s'b \longrightarrow s'(result, out)$ *in cell* σ_1 *has higher priority than rule* $s'a \longrightarrow s'(a, out)$*.*

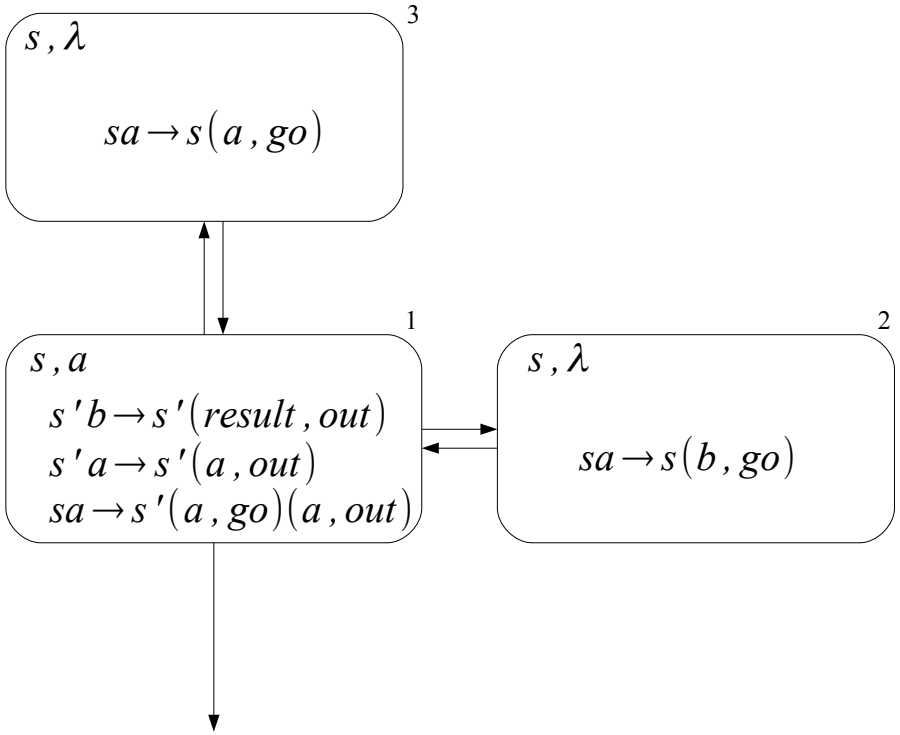


Figure 3.1: An example of a neural-like P system

In the first step, cell σ_1 will apply rule $sa \longrightarrow s'(a, go)(a, out)$ under state s; thus the initial object a in cell σ_1 is consumed and an object a is sent out. One copy of a is sent to cell σ_2, and another copy of a is sent to cell σ_3.

In the second step, cell σ_2 will apply rule $sa \longrightarrow s(b, go)$: this will consume the object a and send an object b to cell σ_1. At the same time, cell σ_3 will apply rule $sa \longrightarrow s(a, go)$: this will consume the object a and send an object a to cell σ_1.

In the third step, both rule $s'b \longrightarrow s'(result, out)$ and rule $s'a \longrightarrow s'(a, out)$ can be applied in cell σ_1. However, since we are assuming that the rule $s'b \longrightarrow s'(result, out)$ has higher priority, this rule is applied to consume the object b and send the string "$result$" out to the environment.

In the fourth step, only the rule $s'a \longrightarrow s'(a, out)$ can be applied, so the remaining object a is consumed and an object a is sent out to the environment. Then no rule can be applied in any cell, and the computation halts.

Thus we get an object a output in the first step, and get one occurrence of the string "$result$" output in the third step with another object a being output in the fourth step, and then the computation halts.

3.3 Recognizer neural-like P systems

In order to study the efficiency of P systems in solving **NP**-complete problems, the key notions of classical *computational complexity theory* have been adapted to membrane computing, and a special class of cell-like P systems was introduced in [PJJC06]: *recognizer P systems*. For tissue P systems, *recognizer tissue P systems* were introduced in [PPJRN08] with the same idea as recognizer cell-like P systems.

The notion of **NP**-completeness is defined in the framework of *decision problems*. Recall that formally a decision problem is a pair (I_X, θ_X), where I_X is a language over a finite alphabet (whose elements are called *instances*) and θ_X is a total Boolean function over I_X.

Recognizer P systems provide a natural framework for solving and studying decision problems within membrane computing, where deciding if an instance of a given problem has an affirmative or negative answer is equivalent to deciding if a string belongs or not to the language associated with the problem. Similarly, *recognizer neural-like P systems* are introduced as below.

Definition 3.3.1 (Recognizer Neural-like P System)**.** *A* recognizer *neural-like P system is a neural-like P system as described in Definition 3.2.1, where:*

- *Every computation halts;*
- *During any computation, either the object* yes *or the object* no *(but not both) must be released into the environment in the last step of the computation.*

We say that a computation of a recognizer neural-like P system is *accepting* if the object yes is emitted by the output cell in the final step.

Definition 3.3.2. *We say that a decision problem* $X = (I_X, \theta_X)$ *is solvable in polynomial time by a family* $\mathbf{\Pi} = \{\Pi(u) | u \in I_X\}$ *of recognizer neural-like P systems if the following conditions all hold:*

- *The family* $\mathbf{\Pi}$ *is* polynomially computable *by Turing machines, that is, there exists a deterministic Turing machine working in polynomial time which constructs the system* $\Pi(u)$ *from* $u \in I_X$*;*
- *There exists a polynomial function* p*, such that for each* $u \in I_X$ *every computation of* $\Pi(u)$ *performs at most* $p(|u|)$ *steps;*
- *(soundness) For each* $u \in I_X$*, if there exists an accepting computation of* $\Pi(u)$ *then* $\theta_X(u) = 1$*;*
- *(completeness) For each* $u \in I_X$*, if* $\theta_X(u) = 1$*, then every computation of* $\Pi(u)$ *is accepting.*

From the soundness and completeness conditions above it follows that every P system $\Pi(u)$ must be confluent, in the following sense: every computation with the same initial configuration must always give the same answer: either all computation are accepting or all computations are not accepting.

We denote by $\mathbf{PMC}^{*}_{NP}$ the set of all decision problems which are solvable in polynomial time by some family of recognizer neural-like P systems.

3.4 Solving the CSP by neural-like P systems

A CSP instance can always be solved using the generate-and-test paradigm where each possible combination of values for the set of variables is systematically generated and then tested to see if it satisfies all the constraints. The combinations that satisfy all the constraints are the solutions of the CSP. The number of combinations considered by this approach is the size of the Cartesian product of all the variable domains.

The solution of the CSP by neural-like P systems proposed here follows a generate-and-test approach in the framework of recognizer neural-like P systems. By broadcasting symbol-impulses to neighboring cells using the replicative mode of communication, and processing strings of symbol-impulses from neighboring cells using the maximal mode of rules application, these neural-like P systems can solve the CSP in polynomial time.

The approach we use consists of the following stages, illustrated in Figure 3.2:

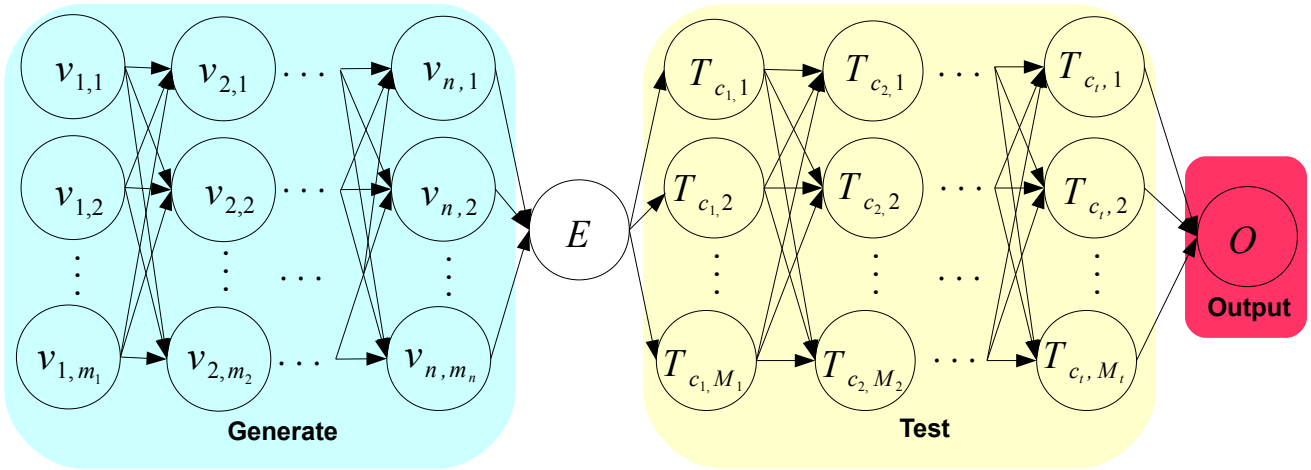


Figure 3.2: Structure of the neural-like P system

Generating Stage: The system generates all the possible combinations of assignments to the variables of the CSP. The strings representing these assignments are constructed using sequential concatenations of possible values for each variable.

Checking Stage: After obtaining all possible combinations encoded as strings of objects, this stage checks whether there is a solution of the CSP by testing these assignments against each constraint in turn. Strings that pass all of these tests and reach the output cell are precisely the strings representing solutions of the CSP instance.

Output Stage: The system sends to the environment any solutions of the CSP that have reached the output cell at the end of the previous two stages, or else outputs an indication that there are no solutions.

Let us consider a CSP instance $\mathcal{P} = (X, D, C)$ with variables $X = \{x_1, x_2, \ldots, x_n\}$, domains $D = \{D_1, D_2, \ldots, D_n\}$, where each set $D_i = \{v_{i,1}, v_{i,2}, \ldots, v_{i,m_i}\}$, and constraints $C = \{C_1, C_2, \ldots, C_t\}$. For each constraint C_j, the allowed assignments to the variables in S_j are tuples of values of length $|S_j|$. We denote the set of assignments allowed by the constraint C_j by $\{T_{C_j,1}, T_{C_j,2}, \ldots, T_{C_j,M_j}\}$ for $1 \leq j \leq t$. The subscript C_j indicates that these are the tuples allowed by the constraint C_j, and the second subscript indicates that there are a total of M_j distinct tuples allowed by this constraint.

We construct a recognizer neural-like P system, $\Pi(\mathcal{P})$, for the CSP instance $\mathcal{P}$, with the following components:

- The set of objects is: $\{yes, no\} \cup \{v_{i,j} | 1 \leq i \leq n, 1 \leq j \leq m_i\}$;
- The set of cells is: $\{\sigma_E, \sigma_O\} \cup \{\sigma_{i,j} | 1 \leq i \leq n, 1 \leq j \leq m_i\} \cup \{\sigma_{C_i,j} | 1 \leq i \leq t, 1 \leq j \leq M_i\}$; where:
  - $\sigma_E = (\{s, s'\}, s, \lambda, \{sz \longrightarrow s'(z, go) | z \neq \lambda\})$;
  - $\sigma_O = (\{s_0, s_1, \ldots, s_{n+t+1}, s'\}, s_0, \lambda, \{s_k\lambda \longrightarrow s_{k+1}\lambda \mid 0 \leq k \leq n+t\} \cup \{s_{n+t+1}z \longrightarrow s'((yes)z, out) \mid z \neq \lambda\} \cup \{s_{n+t+1}\lambda \longrightarrow s'((no), out)\})$;
    Rules of the form $s_{n+t+1}z \longrightarrow s'((yes)z, out)$ have a higher priority than the rule $s_{n+t+1}\lambda \longrightarrow s'((no), out)$.
  - $\sigma_{i,j} = (\{s, s'\}, s, \lambda, \{sz \longrightarrow s'(zv_{i,j}, go)\})$, for $1 \leq i \leq n, 1 \leq j \leq m_i$;
  - $\sigma_{C_i,j} = (\{s, s'\}, s, \lambda, \{sz \longrightarrow s'(z, go) \mid f_{C_i}(z) = T_{C_i,j}\})$, for $1 \leq i \leq t, 1 \leq j \leq M_i$;
    The function $f_{C_i}(z)$ applied to a string $z = g_1 g_2 \ldots g_n$ results in the tuple $f_{C_i}(z) = \{g_k \mid 1 \leq k \leq n, x_k \in S_j\}$. In other words, this function *projects* the assignment represented by the string z down to the variables in S_i, and ensures that the rule can only be applied if this projection gives a combination of values corresponding to the allowed tuple $T_{C_i,j}$.

- The set of synapses is:
  $\{(a_i, a_{i+1}) \mid a_i \in \{(i,1), \ldots, (i,m_i)\}, 1 \leq i < n\} \cup \{(a_n, E) \mid a_n \in \{(n,1), \ldots, (n,m_n)\}$
  $\cup \{(E, b_1) \mid b_1 \in \{(C_1,1), \ldots, (C_1,M_1)\}\} \cup \{(b_i, b_{i+1}) \mid b_i \in \{(C_i,1), \ldots, (C_i,M_i)\}, 1 \leq$
  $i < t\} \cup \{(b_t, O) \mid b_t \in \{(C_t,1), \ldots, (C_t,M_t)\}\}$;

- The output cell is: σ_O.

The recognizer neural-like P system constructed as above is graphically represented in Figure 3.2.

Note that the instance of the CSP is encoded in the structure of the P system (number of cells of each type), and the rules. Thus, there is no specific input cell in the recognizer neural-like P system for solving the CSP.

3.5 An overview of the computation

In this section, we informally describe the operation of the recognizer neural-like P system $\Pi(\mathcal{P})$ described in the previous section.

First, let us start with the generating stage. In this stage, the system generates all the combinations of values for each variable in the form of strings of length n where the ith symbol $g_i = v_{i,j}, 1 \leq i \leq n, 1 \leq j \leq m_i$ (this takes n steps). The strings are constructed starting in cells $\sigma_{1,j}, 1 \leq j \leq m_1$, using sequential concatenations. In each cell $\sigma_{i,j}, 1 \leq i \leq n, 1 \leq j \leq m_i$, the corresponding object $v_{i,j}$ is concatenated and the resulting strings are replicated and passed to the following layer of cells. The generating phase ends with the addition of the last symbol $g_n = v_{n,j}, 1 \leq j \leq m_n$ forming all the $\prod_{i=1}^{n} m_i$ combinations z that reach the cell σ_E. Note that the strings z are manipulated as symbols through the P system. The generating stage takes n steps. The structure of the generative cells is shown in Figure 3.2. Note that the cell σ_E is only defined as a boundary to distinguish between the generating stage and the following checking stage as shown in Figure 3.2.

The checking stage begins from cell σ_E where all the combinations of values for each variable arrive after n steps from the beginning of the computation. The checking stage, which has a very similar structure to the generating stage, consists of t main layers, where each layer is used to check the satisfaction of one constraint of the given CSP. In the ith layer, there are M_i cells, which represent each of the allowed assignments to the variables in S_i, and are connected to each of the cells in the $(i+1)$th layer, as shown in Figure 3.2. All the combinations that are passed on from cell σ_E are filtered in sequence through all the layers of cells in the checking stage. By applying rule $sz \longrightarrow s'(z, go) \mid f_{C_i}(z) = T_{C_i,j}$, cell $\sigma_{C_i,j}$ checks whether the complete assignment specified by the string z matches the corresponding jth allowed assignment tuple of the constraint C_i, for $1 \leq i \leq t, 1 \leq j \leq M_i$. All the allowed assignment tuples for the constraint C_i are checked in

parallel by the ith layer of this stage. The checking stage takes $t+1$ steps to check satisfaction of all the t constraints. After $t+1$ steps, any strings z that reach the output cell σ_O are precisely the solutions of the CSP.

Finally, in the output stage, rule $s_k\lambda \longrightarrow s_{k+1}\lambda \mid 0 \leq k \leq n+t$ makes the output cell σ_O act as a counter to count the steps from the beginning of the computation. If there are any strings in cell σ_O after $(n+t+1)$ steps from the beginning of the computation, the object yes together with the string z is output by applying rule $s_{n+t+1}z \longrightarrow s'((yes)z, out) \mid z \neq \lambda$ in the $(n+t+2)$th step. Otherwise, the object no is output by applying rule $s_{n+t+1}\lambda \longrightarrow s'((no), out)$ in the $(n+t+2)$th step. Note that rule $s_{n+t+1}z \longrightarrow s'((yes)z, out) \mid z \neq \lambda$ has a higher priority than the rule $s_{n+t+1}\lambda \longrightarrow s'((no), out)$. This makes sure the output cell works as described above. The output of the symbol yes together with the string z tells us there is (are) solution(s) for the CSP and the solution(s) is (are) encoded in the string z. However, if there is no solution for the CSP, the object no is sent out indicating that there is no solution.

From this overview of the computation, it is clear that all computations halt in polynomial time ($n+t+2$ steps) and output either the object yes or the object no in the last step of the computation; moreover, the object yes is output precisely when there are one or more solutions to the CSP instance. Hence we have shown that the family $\mathbf{\Pi}$ described above is polynomially bounded, sound and complete. To show that the family $\mathbf{\Pi}$ solves the CSP in polynomial time according to Definition 3.3.2, it only remains to show that it is polynomially computable.

It is easy to check that the components and rules of the system $\Pi(\mathcal{P})$ are precisely specified by the definition of the CSP instance $\mathcal{P}$. Moreover, the necessary resources to build this system are of a polynomial order, as shown below.

We denote by m the maximum size of the domain for any variable, i.e., $m = \max\{m_i \mid 1 \leq i \leq n\}$. Similarly, we denote by M the maximum number of tuples allowed by any constraint, i.e., $M = \max\{M_j \mid 1 \leq j \leq t\}$.

- Size of the alphabet: $\sum_{i=1}^{n} m_i + 2 \in \mathcal{O}(mn)$;
- Number of cells: $\sum_{i=1}^{n} m_i + \sum_{i=1}^{t} M_i + 2 \in \mathcal{O}(mn + Mt)$;
- Initial number of objects: $\sum_{i=1}^{n} m_i \in \mathcal{O}(mn)$;
- Number of rule types: $\sum_{i=1}^{n} m_i + \sum_{i=1}^{t} M_i + 1 + n + t + 2 \in \mathcal{O}(mn + Mt)$;
- Maximal length of rules: $\mathcal{O}(n \log n)$.

All cells apart from cell σ_O have just one rule type. The output cell, σ_O, has $n + t + 2$ rule types. The length of a rule is the number of symbols necessary to write it, considering both its left and right sides. The maximal length of all the rules in $\Pi(\mathcal{P})$ is given by the rule in cell $\sigma_{C_i,j}$ which checks that an assignment string z matches the jth allowed tuple of constraint C_i. This rule specifies a collection of $|S_i|$ indices of positions in z, and a collection of $|S_i|$ values that must occur at these indices. Each index can be specified with $\mathcal{O}(\log n)$ symbols, so the total length of this rule is $\mathcal{O}(n \log n)$.

Therefore, a deterministic Turing machine can build $\Pi(\mathcal{P})$ in polynomial time with respect to n, t, m, M, and hence with respect to the size of the encoding of $\mathcal{P}$.

Hence, by Definition 3.3.2, we have established the following result:

Theorem 3.5.1. *CSP* $\in$ ***PMC***$^*_{NP}$.

Corollary 3.5.2. ***NP*** $\cup$ *co-***NP** $\subseteq$ ***PMC***$^*_{NP}$.

Proof: It suffices to make the following observations: the CSP is well-known to be **NP**-complete [RBW06], CSP $\in$ **PMC**$^*_{NP}$, and this complexity class is closed under polynomial-time reductions and under complement.

3.6 Summary

In this chapter, we have designed a class of neural-like P systems to give a theoretical polynomial-time solution to the constraint satisfaction problem. Even though this is the first proposed use of P systems to solve the general CSP and the method does not rely on generating exponential workspace in polynomial time, it is worth pointing out the impractical nature of this approach. First, the brute-force algorithm employed by the neural-like P systems is only proved to be efficient for the CSP in terms of *time*, and the number of objects generated by this class of neural-like P systems is exponentially large, making the implementation of such a system rather impractical. Second, the use of rewrite rules in the neural-like P systems for CSP solving relies on sophisticated pattern matching. This requires an exponential number of specific pattern matching rules in the neural-like P systems which also impedes its practical implementation.

In the next chapter, we will explore a more practical use of neural-like P systems by designing a simple class of neural-like P systems with probabilistic features to solve the distributed MIS selection problem efficiently.

CHAPTER 4

Simple neural-like P systems for distributed MIS selection

In this chapter, we design a class of simple neural-like P systems—neural-like probabilistic P systems—which implements the improved version of the MIS algorithm originally inspired by the fly's nervous system proposed in [AABJ$^+$11]. Hence we implement a biologically-inspired algorithm using a class of biologically-inspired computing models. This class of simple neural-like P systems for distributed MIS selection illustrates a new method for using membrane computing, and provides a new bridge between membrane computing and distributed computing.

We first give the definition and a simple example of neural-like probabilistic P systems. In the following section, we design a class of neural-like probabilistic P systems for solving the distributed MIS selection problem. Finally, we give a short summary of this chapter.

4.1 Neural-like probabilistic P systems

In this section we introduce the basic definitions of neural-like probabilistic P systems based on the definition of neural-like P systems introduced in Definition 3.2.1. Here, we add probabilistic features to this class of P systems, as has been done with other types of P systems, to construct neural-like probabilistic P systems. The general definition is as follows:

Definition 4.1.1 (Neural-like Probabilistic P System)**.** *A neural-like probabilistic P system (NPP) of* degree $n \geq 1$ *is a tuple* $(O, \sigma_1, \ldots, \sigma_n, syn, i_{out})$*, where:*

- O *is a finite alphabet, and its elements are called objects;*
- $\sigma_1, \ldots, \sigma_n$ *are cells, each of the form* $\sigma_i = (Q_i, s_{i,0}, w_{i,0}, R_i), 1 \leq i \leq n$*, where:*

1) *Q_i is a finite set of possible states for the cell σ_i;*

2) *$s_{i,0} \in Q_i$ is the initial state of the cell σ_i;*

3) *$w_{i,0} \in O^*$ represents the initial multiset of objects in the cell σ_i;*

4) *R_i is a finite set of rules of the form $sw \xrightarrow{p} s'xy_{go}z_{out}$, where $0 \leq p \leq 1$ indicates the probability of the rule being applied under appropriate conditions (if $p = 1$ it is usually omitted). The probability p may vary as a function of the running time (number of steps) t. The symbols $s, s' \in Q_i, w, x \in O^*$, $y_{go} \in (O^* \times \{go\})$ and $z_{out} \in (O^* \times \{out\})$, with the restriction that $z_{out} = \lambda$ (the empty multiset) for all $i \in \{1, 2, \ldots, n\}$ not contained in i_{out}.*

- *$syn \subseteq \{1, 2, \ldots, n\} \times \{1, 2, \ldots, n\}$ specifies a set of synapses connecting cells;*
- *$i_{out} \subseteq \{1, 2, \ldots, n\}$ specifies a set of output cells.*

A *configuration* of a neural-like probabilistic P system at step t, $\langle syn, (s_{1,t}, w_{1,t}), (s_{2,t}, w_{2,t}), \ldots, (s_{n,t}, w_{n,t})\rangle$, consists of a cell structure (a set of cells and the synapses connecting them), the states of all the cells and the multisets of objects present inside each of the cells. The *initial* configuration, $\langle syn, (s_{1,0}, w_{1,0}), (s_{2,0}, w_{2,0}), \ldots, (s_{n,0}, w_{n,0})\rangle$, consists of the cell structure, the initial states of all the cells and the initial multisets of objects in all cells.

A rule in a neural-like probabilistic P system is applied similarly to the way rules are applied in the neural-like P systems introduced in Definition 3.2.1: For each cell σ_i the set of rules, R_i, contains rules of the form $sw \xrightarrow{p} s'xy_{go}z_{out}$, where s, s' are states and w, y, z are multisets of objects (described by strings). A rule $sw \xrightarrow{p} s'xy_{go}z_{out}$ will be called *applicable* in a cell σ if σ is in state s and the multiset represented by w is contained in the multiset of objects in σ. When such a rule is applied to a multiset w in state s, the multiset w is replaced by x which stays in the current cell, y_{go} which goes to neighbouring cells, and z_{out} which is sent out of the system. (The multiset z_{out} is required to be empty except possibly in output cells.)

In neural-like probabilistic P systems each rule is associated with a fixed priority level, and a probability value p between 0 and 1, which may vary as a function of time. In each cell, at each step, the applicable rules are considered in decreasing order of priority. (If there is more than one applicable rule at the priority level currently being considered, then one of them is chosen non-deterministically.) For each rule considered, it is decided by some stochastic process whether or not to use that rule, based on its associated probability value: a rule with associated probability value p is used with probability p. If this stochastic process decides to use the rule, then it is applied in parallel as many times as possible, given the current multiset of objects in the cell, and the computation in this cell at this step is then finished. If the process decides not to use this rule, then the next (lower) priority level is considered. If in the end, after considering all priority levels, no rule has

been applied, then the configuration of the cell remains the same in this step. Note that either no rules or exactly one rule is used at each step in each cell, but when a rule is used it may be applied many times in parallel in order to process as many of the objects in the cell as possible. Because rules with higher priority are considered first, we call this way of using rules *priority-based*.

As usual for standard P systems, we assume the existence of a global clock which marks the timing of steps (single transitions) for the whole system.

A single transition from a configuration to a new one, i.e., from $\langle syn, (s_{1,t}, w_{1,t}), (s_{2,t}, w_{2,t}), \ldots, (s_{n,t}, w_{n,t})\rangle$ to $\langle syn, (s_{1,t+1}, w_{1,t+1}), (s_{2,t+1}, w_{2,t+1}), \ldots, (s_{n,t+1}, w_{n,t+1})\rangle$, is performed by applying the rules in each cell of the system in parallel. A cell remains unchanged during the transition if and only if no rules can be applied to it. The transitions of neural-like probabilistic P systems give rise to a markov chain whose states are the corresponding configurations. A sequence of transitions, starting from the initial configuration $\langle syn, (s_{1,0}, w_{1,0}), (s_{2,0}, w_{2,0}), \ldots, (s_{n,0}, w_{n,0})\rangle$ to the configuration $\langle syn, (s_{1,t}, w_{1,t}), (s_{2,t}, w_{2,t}), \ldots, (s_{n,t}, w_{n,t})\rangle$ after t steps, is called a *computation* after t steps. A computation is said to be halting if it reaches a configuration where no rule can be applied anywhere in the system.

Owing to the use of probabilities in neural-like probabilistic P systems, and non-determinism in the choice of rules, one can, in general, get many possible computations starting from the same initial configuration, and thus many possible results.

We now present a simple example of a neural-like probabilistic P system in order to clarify the definitions and to illustrate the way these systems work.

Example 4.1.2. *Our example of an NPP is the construct* $(O, \sigma_1, \sigma_2, syn, i_{out})$*, where:*

- $O = \{a, b\}$*;*
- $\sigma_1 = (\{s\}, s, a, \{sb \longrightarrow s(b, out), sa \longrightarrow s(a, go)(a, out)\})$*;*
- $\sigma_2 = (\{s\}, s, \lambda, \{sa \xrightarrow{\frac{1}{2}} s(b, go)\})$*;*
- $syn = \{(1, 2), (2, 1)\}$*;*
- $i_{out} = \{1\}$.

This NPP is graphically represented in Figure 4.1, where ovals represent the cells (these ovals indicate the initial state, the initial multiset of objects, and the set of rules, and are labeled $1, 2, \ldots, n$*), arrows between cells indicate the synapses, and there is an arrow leaving from the output cells.*

We briefly describe how this NPP works (assuming priority-based rule application as described above, and replicative communication). In the first step, cell 1 will apply rule $sa \longrightarrow s(a, go)(a, out)$*. Thus the initial object* a *in cell 1 is consumed, a single object of type* a *is sent out, and a single object*

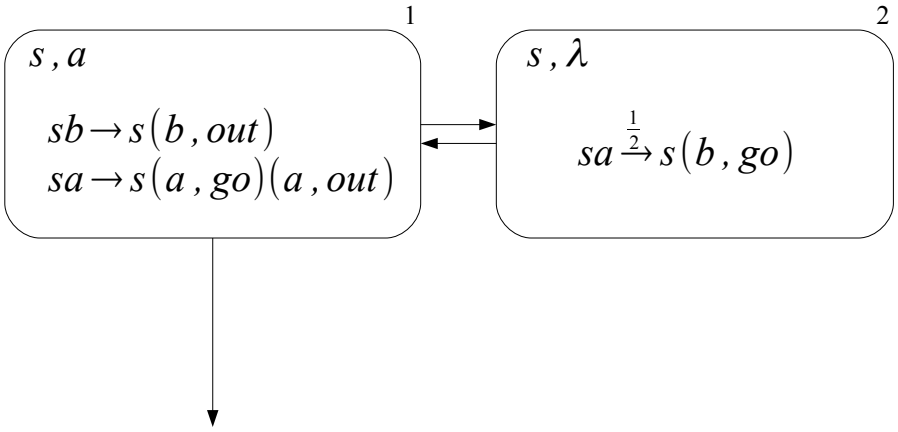


Figure 4.1: An example of a neural-like probabilistic P system

of type a is sent to cell 2. In the following steps, cell 2 will use a stochastic process to decide whether to apply rule $sa \xrightarrow{\frac{1}{2}} s(b, go)$ with probability $\frac{1}{2}$. If this rule is successfully applied it will consume the object a and send an object of type b to cell 1. Once this has happened, rule $sb \longrightarrow s(b, out)$ will be applied in cell 1 at the next step to consume the object b and send an object of type b out to the environment. After that no rule is applicable in either cell 1 or cell 2, and the computation halts. Thus we get an a output in the first step, and may get one occurrence of b output at any point from step 3 onwards (with probability $\frac{1}{2}$ in each step), at which point the computation halts.

4.2 Neural-like probabilistic P systems for solving the MIS selection problem

As mentioned at the beginning of this chapter, it was noticed by Afek et al. that the requirements for sensory organ precursor selection in the developing brain of a fly are formally similar to the requirements for MIS selection in a given graph. They used this observation to design a new form of randomised distributed algorithm for MIS selection [AAB$^+$11]. In this new distributed algorithm the communication between cells is as simple as possible – it consists of single bits.

As originally presented [AAB$^+$11], the new distributed algorithm needed to know the total number of nodes of the graph and the maximum degree of the graph in order to choose appropriate probability values. However, the algorithm was later improved by Afek et al. without sacrificing its running time or message complexity [AABJ$^+$11]. The improved version as shown in Table 4.1 requires no knowledge about the overall size of the graph or the number of active neighbours at a node. The running time of the new algorithm is slightly greater than Luby's algorithm, but this bio-

inspired algorithm is still very efficient, and is more robust and applicable in many contexts because it overcomes some drawbacks of Luby's algorithm.

Table 4.1: The bio-inspired MIS selection algorithm at each node (adapted from [AABJ$^+$11])

Local variables: k : phase counter;
i : time step counter;
p : local probability value;
TRYING : Boolean flag, initialised to FALSE.

1. Set $k \leftarrow 0$;
2. **while** active, at each time step **do**
3. Set $i \leftarrow 0$;
4. Set $k \leftarrow k + 1$;
5. **while** $i \leq k$ **do**
6. Set $p \leftarrow 1/2^i$;
7. Set $i \leftarrow i + 1$;
8. *FIRST EXCHANGE*
9. With probability p, set TRYING $\leftarrow$ TRUE and **send** signal to all neighbours;
10. **Receive** any signals sent by neighbours;
11. **if** any signal was received **then**
12. TRYING $\leftarrow$ FALSE
13. *SECOND EXCHANGE*
14. **if** TRYING **then**
15. **Send** signal to all neighbours;
16. Join the MIS and terminate (become inactive).
17. **Receive** any signals sent by neighbours;
18. **if** any signal was received **then**
19. Terminate (become inactive)

The key idea of the new algorithm is to vary over time the probability that nodes will attempt to join the MIS. This was based on the observation that, during the development of the fly brain, the probability that any cell will self-select as an SOP varies not as a function of the number of connections to other cells, as in Luby's algorithm, but as a function of time.

We now present a class of neural-like probabilistic P systems which are designed to implement the improved synchronous randomised distributed MIS algorithm devised by Afek et al. [AABJ$^+$11].

Definition 4.2.1. *We define a class of neural-like probabilistic P systems for distributed MIS selection on undirected graphs, as follows. For each undirected graph G with n nodes we set $\Pi_{MIS}(G) = (O, \sigma_1, \ldots, \sigma_n, syn, i_{out})$, where:*

- $O = \{b\}$;

- $\sigma_1, \ldots, \sigma_n$ *are identical cells corresponding to the nodes of* G*, each of the form* $\sigma_i = (Q_i, s_{i,0}, w_{i,0}, R_i)$*, where:*

  1) $Q_i = \{s_0, s_1, s_2, s_3, s_4\}$*;*

  2) $s_{i,0} = s_0$ *is the initial state of each cell;*

  3) $w_{i,0} = \lambda$ *(which means each cell is initially empty);*

  4) $R_i = \{s_0 b \longrightarrow s_3, s_0 \xrightarrow{p} s_2(b, go), s_0 \longrightarrow s_1, s_1 b \longrightarrow s_0, s_1 \longrightarrow s_0, s_2 b \longrightarrow s_0, s_2 \longrightarrow s_4(b, go)\}$*. The priority of these rules is as defined below:*

  a) rules applicable in s_0*:* $s_0 b \longrightarrow s_3 > s_0 \xrightarrow{p} s_2(b, go) > s_0 \longrightarrow s_1$*;*

  b) rules applicable in s_1*:* $s_1 b \longrightarrow s_0 > s_1 \longrightarrow s_0$*;*

  c) rules applicable in s_2*:* $s_2 b \longrightarrow s_0 > s_2 \longrightarrow s_4(b, go)$*.*

- $syn \subseteq \{1, 2, \ldots, n\} \times \{1, 2, \ldots, n\}$ *are synapses connecting cells, which correspond to the edges of* G *(there exists a pair of synapses* $\{(i, j), (j, i)\}$ *if and only if there exists an edge connecting two nodes* i *and* j *in the graph* G*);*

- $i_{out} = \{1, 2, \ldots, n\}$*, so all of the cells are output cells;*

We specify how the probability p in the rule $s_0 \xrightarrow{p} s_1(b, go)$ varies with time. Each pair of consecutive steps is considered to be one *time step* of the computation. Following the scheme specified in [AABJ$^+$11] (see Table 4.1), we divide the computation into *phases*, numbered $1, 2, 3, \ldots$. Each phase k consists of $k + 1$ time steps (i.e., $2k + 2$ steps). The value of the probability p varies as follows: during each phase k the value of p is 1 initially, and gets halved after each successive time step in phase k. Thus, the value of p during the computation will take the following values in successive time steps: $\underline{1, \frac{1}{2}}, \underline{1, \frac{1}{2}, \frac{1}{4}}, \underline{1, \frac{1}{2}, \frac{1}{4}, \frac{1}{8}}, \underline{1, \frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{16}}, \ldots$ (where the underlinings indicate the phases).

A single cell of the neural-like P system for distributed MIS selection is shown in Figure 4.2. Each such cell represents one node of the graph. Cells are connected by synapses representing the edges connecting the nodes in the graph.

Each cell is identical and has five states.

- s_0 is the initial state;

- s_1 is a transitional state indicating that the cell did not try to join the MIS on the previous step;

- s_2 is another transitional state indicating that the cell did try to join the MIS on the previous step;

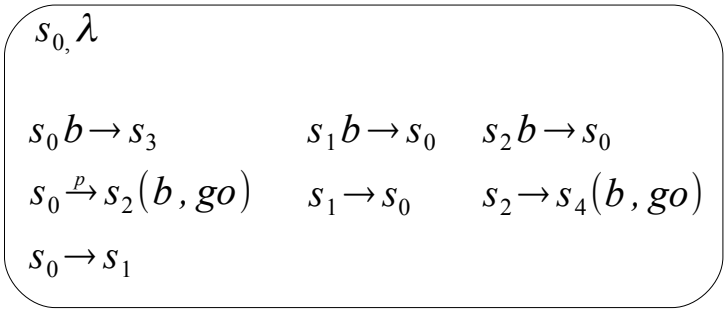


Figure 4.2: A single cell of the neural-like P system for distributed MIS selection

- s_3 is a terminal state, indicating that the cell is connected to a cell which is a member of the MIS;
- s_4 is a terminal state, indicating that the cell is a member of the MIS.

Cells in state s_0, s_1 and s_2 are called active cells. Both s_3 and s_4 are terminal states, and cells in these two states are inactive and play no further part in the computation.

Initially, each cell is in its initial state s_0 and does not contain any object.

At the start of the first step of the computation ($t = 0$), the highest priority applicable rule is the rule $s_0 \xrightarrow{p} s_2(b, go)$. Hence, this rule will be applied in each cell with probability p, which means that, with probability p, each cell sends an object b to all its neighbours and changes its state from s_0 to s_2. In any cell where the rule $s_0 \xrightarrow{p} s_2(b, go)$ is not applied, the rule $s_0 \longrightarrow s_1$, which has a lower priority, will be applied to change its state from s_0 to s_1. Hence, after the first step of the computation, all the cells are either in state s_1 or state s_2.

In the second step ($t = 1$), all the cells begin either in state s_1 or state s_2. The cells in state s_1 will apply either the rule $s_1 b \longrightarrow s_0$ (if they received one or more objects b in the previous step) or the rule $s_1 \longrightarrow s_0$ (if they did not receive any object b in the previous step). Hence these cells will all return to state s_0. Meanwhile, cells in state s_2 which received one or more copies of object b at the previous step will return to the initial state s_0, with no objects, by applying the rule $s_2 b \longrightarrow s_0$. Moreover, cells in state s_2 which did not receive object b during the previous step will join the MIS and broadcast object b to their neighbours by applying the rule $s_2 \longrightarrow s_4(b, go)$. Hence, after the second step of the computation, all the cells are either in state s_0 or state s_4.

In the third step ($t = 2$), all the cells begin either in state s_0 or state s_4. Cells in state s_4 are inactive, they have no applicable rules, so they no longer need to be considered in the computation. Cells in state s_0 will repeat the computation of the first step, except that some of them may have

received object b in the previous step from cells joining the MIS. Cells in state s_0 that received one or more copies of object b in the previous step will apply the rule $s_0 b \longrightarrow s_3$ to transfer to state s_3 and thus become inactive.

For the rest of the computation, all the even-step (t is even) computations will repeat the computation of the third step ($t = 2$); all the odd-step (t is odd) computations will repeat the computation of the second step ($t = 1$), until all nodes are inactive and the computation halts. Hence the above process repeats again and again to eventually select all the members of an MIS. Note that the application of rules obeys the priority order specified in Definition 4.2.1. The computation halts when there is no applicable rule in any cell.

A brief explanation of each of the rules is given in Table 4.2.

Table 4.2: State transfer and rules used in an NPP for distributed MIS selection

State transfer	Rules applied	Meaning
$s_0 \rightarrow s_3$	$s_0 b \longrightarrow s_3$	if the cell contains b, transfer from s_0 to s_3 and clear b
$s_0 \rightarrow s_2$	$s_0 \xrightarrow{p} s_2(b, go)$	with probability p, transfer from s_0 to s_2 and broadcast b
$s_0 \rightarrow s_1$	$s_0 \longrightarrow s_1$	transfer from s_0 to s_1
$s_1 \rightarrow s_0$	$s_1 b \longrightarrow s_0$	if the cell contains b, transfer from s_1 to s_0 and clear b
$s_1 \rightarrow s_0$	$s_1 \longrightarrow s_0$	transfer from s_1 to s_0
$s_2 \rightarrow s_0$	$s_2 b \longrightarrow s_0$	if the cell contains b, transfer from s_2 to s_0 and clear b
$s_2 \rightarrow s_4$	$s_2 \longrightarrow s_4(b, go)$	transfer from s_2 to s_4 and broadcast b

The computation of each individual cell can also be described by a finite state automaton, as shown in Figure 4.3, where nodes of the automaton represent different states, and the conditions to be fulfilled to transfer from one state to another are specified on the transition arrows.

To prove the correctness of the class of NPPs described in Definition 4.2.1 for solving the distributed MIS problem we will establish that when the computation of an NPP halts, no two neighbouring cells are in the MIS (i.e., in state s_4) and that each cell is either a member of the MIS (in state s_4) or connected to a cell in the MIS (in state s_3).

Lemma 4.2.2. *No two cells that join the MIS are adjacent in the graph.*

Proof. A cell u in s_0 can only reach state s_4 (i.e., join the MIS) if it transfers to state s_2 and then to state s_4. To make these two transitions it must broadcast object b at some even-numbered step t, and not receive object b at that step. This means that none of its neighbours tries to join the MIS at the same step, and hence all active neighbours must transfer to state s_1 at step t.

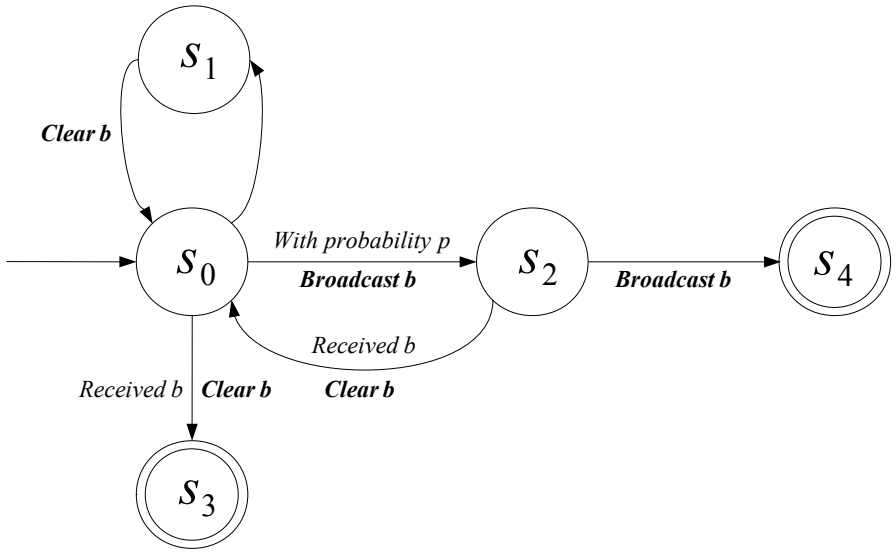


Figure 4.3: Automaton description of an NPP for distributed MIS selection

When cell u transfers to state s_4 from state s_2, and thus joins the MIS, all its neighbours will receive message b at the end of step $t + 1$, and hence will transfer to state s_3 in the next step. Thus cell u joins the MIS only when it broadcasts object b (without getting b from its neighbours in the previous step), and none of its neighbours broadcasts b at the same step. So two neighbouring cells cannot join the MIS at the same step, and when the earlier of them joins, the other one will then transfer to state s_3 to become inactive and will never join the MIS. □

Lemma 4.2.3. *If a cell joins the MIS, then all its neighbouring cells will become inactive.*

Proof. By Lemma 1 we know that none of u's neighbours is in the MIS. When cell u joins the MIS it broadcasts to all its neighbours message b on an odd-numbered step to claim that it has become a member of the MIS, then all its neighbours receiving the message b will transfer to state s_3 to become inactive and their computation halts. □

Lemma 4.2.4. *If the computation of a cell halts, and it did not join the MIS, then it is connected to a cell in the MIS.*

Proof. The computation of cell u will halt only when it is either in state s_4 (signifying that it has joined the MIS) or in state s_3 (because it received a message b from a neighbouring cell that has just joined the MIS). □

We have specified how the probability p in the rule $s_0 \xrightarrow{p} s_1(b, go)$ varies with time. Thus the value of p during the computation will take the following values in successive time steps: $\underline{1, \frac{1}{2}}, \underline{1, \frac{1}{2}, \frac{1}{4}}, \underline{1, \frac{1}{2}, \frac{1}{4}, \frac{1}{8}}, \underline{1, \frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{16}}, \ldots$ (where the underlinings indicate the phases). An analysis of the expected running time of this algorithm for solving the distributed MIS is given in [AABJ$^+$11]. They show that in every phase with $\log n$ or more time steps, at least some constant fraction of the edges of the graph are expected to become inactive (i.e., to link nodes that become inactive). To finish the algorithm, $O(\log n)$ phases with more than $\log n$ time steps are needed in expectation. Hence they derive an upper bound of $O(\log^2 n)$ on the expected total running time of the algorithm.

Theorem 4.2.5. *Let G be any graph with n nodes, and let $\Pi_{MIS}(G)$ be the corresponding NPP, as described in Definition 4.2.1, where the value of probability p in rule $s_0 \xrightarrow{p} s_1(b, go)$ varies as follows: in each phase k, for $k = 1, 2, 3, \ldots$, consisting of $k+1$ time steps, the value of probability p is 1 initially, and is halved after each successive time step. The expected total number of time steps required to compute an MIS is $O(\log^2 n)$.*

Proof. The fact that any halting computation of $\Pi_{MIS}(G)$ will correctly identify an MIS of G has been established in Lemmas 4.2.2, 4.2.3 and 4.2.4. Moreover, $\Pi_{MIS}(G)$ implements the randomised algorithm for selecting an MIS specified in [AABJ$^+$11]. It is shown in [AABJ$^+$11] that if the probability values are chosen in the way specified in the theorem, then the expected total number of time steps until the algorithm halts is $O(\log^2 n)$. □

4.3 Summary

In this chapter we have designed a class of neural-like probabilistic P systems to solve one of the most fundamental distributed computational problems: the distributed maximal independent set selection problem. Our neural-like probabilistic P systems contain simple identical cells and use only one-bit messages to complete this distributed computational task. Moreover, the individual processors (cells) need no information about the size of the graph G, and use a simple and regular set of probability values (inverse powers of 2), which makes the class of NPP models proposed in this paper easily implementable in very simple hardware.

By solving the distributed MIS selection problem we have shown that neural-like probabilistic P systems have great potential for solving distributed computational problems simply and efficiently. Can the algorithmic mechanism employed by the neural-like probabilistic P systems for distributed MIS selection be adapted to solve other distributed computational problems? Chapter 5 gives an affirmative answer by extending the algorithm in Table 4.1 to solve the distributed greedy colouring problem.

Now consider a class of algorithms similar to the one described in Table 4.1 and implemented in the neural-like probabilistic P systems above where each node runs through the same fixed preset sequence of probability values, and does not adjust these to take into account the behaviour of other nodes. We refer to this class of algorithms as "MIS selection with global probability values".

The next result, due to Alex Scott [SJX13], describes an explicit family of graphs with $O(n)$ vertices, for which *any* such algorithm takes at least $\Omega(\log^2 n)$ time steps, no matter what sequence of probability values is used.

Theorem 4.3.1. *There is a constant $\kappa > 0$ such that the following holds. Let G be the graph consisting of $n^{1/3}$ disjoint copies of the complete graph K_d, for each $d = 1, \ldots, n^{1/3}$. Then with high probability, any MIS selection algorithm with global probability values running on G does not terminate within $\kappa \log^2 n$ time steps.*

Does this mean that the simple mechanism used by biological cells for "fine-grained" pattern formation has an inherently lower efficiency compared with carefully engineered algorithms such as Luby's algorithm which requires only $O(\log n)$ time steps? Chapter 6 will answer this question by looking more closely at the mechanism of "fine-grained" pattern formation during cell development.

CHAPTER 5

A cellular computing approach to greedy colouring

In this chapter, we extend the bio-inspired MIS selection approach to solve the distributed greedy colouring problem with a very simple and similar mechanism. More specifically, we consider extremely restrictive conditions for greedy colouring: each vertex in the graph hosts a processor. The communication is synchronous and proceeds in discrete time steps. The processors have minimal knowledge about the graph, and the only messages they can send at each time step are single colour values, which are broadcast to all of their neighbours.

Previous algorithms for distributed greedy colouring generally assume that some local or global properties of the graph are known to each of the vertices, and often require complex message exchanges (see Section 2.3.3). However, our new approach to distributed greedy colouring inspired by the local pattern formation mechanism in animal development avoids these drawbacks. Working strictly within Linial's computation model, we impose the further constraint that our algorithms use only very simple processing elements, with minimal knowledge of the graph structure, and communicate only simple messages corresponding to the set of colour values used in the final colouring. Moreover, our algorithms communicate by broadcasting their messages to all neighbours, and do not need to send different messages to different neighbours. Such algorithms are likely to be much easier to implement and deploy in a wide-range of computing contexts, including wireless communication networks [CK10].

We show that even under such restrictive conditions we can construct a randomised algorithmic scheme that computes a greedy colouring distributively. We consider two different variants of this algorithmic scheme. In the first version, Algorithm 1 (Table 5.2), the processors at each vertex of the graph have no information at all about the graph, and use a predetermined sequence of probability values to decide whether or not to send a message at each time step. This version computes a greedy

colouring after an expected number of time steps which is $O(\Delta^2 \log^2 n)$, where n is the number of vertices and Δ is the maximum degree of the graph. The expected number of messages sent by each node is $O(\Delta \log n)$.

In the second version, Algorithm 2 (Table 5.3), the processors at each vertex have information about the total number n of the vertices of the graph, and use this information to select a tailored sequence of probability values. This version computes a greedy colouring after an expected number of time steps which is $O(\Delta \log^2 n)$. The expected number of messages sent by each node is again $O(\Delta \log n)$.

The chapter is organized as follows: We first mention some preliminaries. Then we present our general randomised algorithmic scheme for distributed greedy colouring and prove its correctness. In the next section, we consider two specific variants of this algorithmic scheme: Algorithm 1, which assumes that each vertex has no information about the graph and Algorithm 2, which assumes that each vertex knows only the total number of vertices. We analyse the time and message complexities for both variants. Finally, we give a short summary of this chapter.

5.1 Preliminaries

Given an undirected graph $G = (V, E)$, the *neighbourhood* of each vertex $v \in V$ is defined to be the set $\Gamma(v) = \{u : \{u, v\} \in E\}$ and the *degree* of each vertex v is defined to be the number $\deg_G(v) = |\Gamma(v)|$. We recall that the maximum degree of the graph G is defined to be the maximum value of the degree over all vertices of G, which is denoted by $\Delta = \max_{v \in V}\{\deg_G(v)\}$. The number of vertices of G is $|V|$ and will usually be denoted by n. An event on G is said to occur *with high probability*, if it occurs with probability $1 - o(1)$. We will write $\log a$ for the natural logarithm of a, and $\log_b a$ for the logarithm of a to the base b.

5.2 Basic algorithmic scheme for distributed greedy colouring

For an arbitrary n-vertex graph $G = (V, E)$, we consider a synchronous communication network modelling the graph G, where each vertex is represented by a processor and each edge is represented by a bi-directional communication channel connecting the corresponding processors. The only messages that we allow represent individual colour values chosen from some ordered set of possible colour values.

Our basic algorithmic scheme at each processor is shown in Table 5.1.

The basic idea of this algorithmic scheme is to vary over time the probability that each vertex will attempt to choose a colour. If two neighbouring vertices attempt to choose the same colour, then they will both abandon choosing that colour in that time step. However, if a vertex attempts to

Table 5.1: Basic algorithmic scheme for distributed greedy colouring at each processor

Local variables: p : local probability value;
TRYING : Boolean flag, initialised to FALSE;
S : Set of forbidden colours, i.e., those taken by neighbours, initialised to $\emptyset$.

1. Set $S \leftarrow \emptyset$;
2. **while** active, at each time step **do**
3. Set probability value p;
4. *FIRST EXCHANGE*
5. Choose the smallest available colour c that is not in S;
6. With probability p, set TRYING $\leftarrow$ TRUE and **send** c to all neighbours;
7. **Receive** any colour signals sent by neighbours;
8. **if** any neighbour sent colour c **then**
9. TRYING $\leftarrow$ FALSE
10. *SECOND EXCHANGE*
11. **if** TRYING **then**
12. **Send** c to all neighbours;
13. Assign colour c to this node and terminate (become inactive).
14. **Receive** any colour signals sent by neighbours and add them to S.

choose a colour and none of its neighbours attempt to choose the same colour at that time step, then it will be successfully assigned, and all neighbours will be notified.

The computation at each time step consists of two rounds of message exchanges: in the first exchange, each vertex adds any colours broadcast at the end of the previous time step to the set of forbidden colours S. It then selects the first available colour that is not in the set S, and broadcasts this colour with probability p. In the second exchange the vertex receives all the messages from its neighbours and checks whether any neighbour has also broadcast the same colour. If not, then the vertex successfully assigns itself that colour, broadcasts the colour to all its neighbours to forbid their use of that colour in future, and becomes inactive. It then plays no further role in the computation.

Note that the colours received by a vertex in the first exchange are the colours broadcast by any of its neighbours that got assigned a colour in the preceding second exchange. This ensures the colours in the set S record the colours of the already-coloured neighbouring vertices of each vertex.

We first prove the correctness of this general algorithmic scheme for computing a greedy colouring.

Lemma 5.2.1. *No two vertices that get assigned the same colour value at the same time step are adjacent in the graph.*

Proof. Each time step of the computation consists of two exchanges. A vertex can only be assigned

a colour c if it broadcasts the colour c in the first exchange, and does not receive c in the second exchange. This means that none of its neighbouring vertices broadcasts the same colour c in the first exchange, and hence all neighbours must either not broadcast any colour, or alternatively broadcast colours that are different from c in the first exchange. Hence none of its neighbours can also be assigned colour c at the same time step. □

Lemma 5.2.2. *A colour value that is assigned to any vertex must be the lowest colour value that is different from any of its neighbouring vertices which are already coloured.*

Proof. If a vertex v gets assigned the colour c at some time step, then it must have chosen and broadcast c in the first exchange. This means that c is the lowest colour value that is not in the set S at the beginning of that time step.

If any neighbour of v has already been assigned a colour value, at some previous time step, then it must have broadcast that value in the second exchange of that time step, and hence that value will have been added to the set S.

Thus, when a vertex is assigned a colour value, that colour value is always the lowest colour value that is different from the colour values assigned to any of its neighbouring vertices which are already coloured. □

Theorem 5.2.3. *If the algorithmic scheme in Table 5.1 is run on the vertices of any graph G, and all vertices become inactive, then the colour values assigned to each of the vertices define a greedy colouring of G.*

Proof. By Lemma 5.2.1 and Lemma 5.2.2, when every vertex in G is assigned a colour value, no two neighbouring vertices are assigned the same colour value, so these colour values constitute a valid colouring of the graph G.

Furthermore, each colour value assigned to a vertex cannot be replaced by a lower value without violating the property that neighbouring vertices are assigned different colours. Thus, these colour values form a Grundy colouring, and hence a greedy colouring, of the graph G. □

We have now described our basic algorithmic scheme for distributed greedy colouring, and given a correctness proof. To analyse its performance, in terms of the total number of time steps required and the total number of messages sent, it is necessary to indicate how the probability value p is set at each vertex in each time step. We will consider two kinds of approaches to updating the value of p in the next two sections: in one of these the value of p is set without any knowledge of the graph, and in the other it is adjusted based on a knowledge of the total number of vertices in the graph.

In each case, our analysis of the running time will depend on the strong link between the greedy colouring problem and the problem of calculating a maximal independent set (see Section 2.3.2).

A strong connection between maximal independent sets and greedy colourings is captured by the following result.

Lemma 5.2.4. *In any greedy colouring of a graph G, the set of vertices with colour value c is a maximal independent set in the subgraph of G induced by the vertices with colour value c or higher.*

Proof. Given a greedy colouring of G, the vertices with colour value c are non-adjacent to each other. Moreover, vertices with colour value $c' > c$ must have at least one neighbour with colour value c since the colouring is greedy. Hence these vertices form a maximal independent set in the graph induced by the vertices with colour value c or higher. □

5.3 Two algorithms for distributed greedy colouring

In this section, we propose and analyse two variants of distributed greedy colouring algorithms based on the algorithmic scheme described in Section 5.2.

5.3.1 Algorithm 1: distributed GC without any graph knowledge

Algorithm 1 is described in Table 5.2. It uses a similar approach to the algorithm for distributed MIS selection described in Table 4.1. Each vertex (processor) in the graph has a variable k with initial value of 0 indicating the *phase* of the computation. Each phase k consists of k time steps, indexed by the variable i, which is set to 0 at the start of each phase, and incremented at each time step. The probability p that each vertex attempts to choose a colour is set to $1/2^i$. Hence the value of the probability p varies as follows: during each phase k the value of p is 1 initially, and then gets halved after each successive time step until the end of the current phase. Thus, the value of p during the computation will take the following values in successive time steps: $\underline{1}, \underline{1, \frac{1}{2}}, \underline{1, \frac{1}{2}, \frac{1}{4}}, \underline{1, \frac{1}{2}, \frac{1}{4}, \frac{1}{8}}, \underline{1, \frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{16}}, \ldots$ (where the underlinings indicate the phases starting from phase $k = 1$). The probability values selected in the first 10 phases of Algorithm 1 are shown in Figure 5.1a.

We now analyse the running time of Algorithm 1, as follows:

Theorem 5.3.1. *Given any graph G with n vertices and maximum degree Δ, the expected total number of time steps required by Algorithm 1 to compute a greedy colouring of G is $O(\Delta^2 \log^2 n)$.*

Proof. The fact that any complete computation of Algorithm 1 will correctly output a greedy colouring of G was established in Theorem 5.2.3.

By Lemma 5.2.4, we know that the vertices assigned colour value c constitute a maximal independent set in the subgraph induced by the vertices assigned colour value c or higher. We denote the process that results in assigning colour c to some vertices to form this maximal independent set by $MIS(c)$. We denote by $T(c)$ the set of time steps which attempt to assign colour c to some vertices

Table 5.2: Algorithm 1 at each processor

Local variables: k : phase counter;
i : time step counter;
p : local probability value;
TRYING : Boolean flag, initialised to FALSE;
S : Set of forbidden colours, i.e., those taken by neighbours, initialised to $\emptyset$.

1. Set $k \leftarrow 0$;
2. Set $S \leftarrow \emptyset$;
3. **while** active, at each time step **do**
4. Set $i \leftarrow 0$;
5. Set $k \leftarrow k + 1$;
6. **while** $i < k$ **do**
7. Set probability $p \leftarrow 1/2^i$;
8. Set $i \leftarrow i + 1$;
9. *FIRST EXCHANGE* (As in Table 5.1);
10. *SECOND EXCHANGE* (As in Table 5.1).

to form $MIS(c)$. In other words, $T(c)$ represents the set of time steps where some vertices attempt to choose c as their smallest available colour.

The processes $MIS(c_1), MIS(c_2), MIS(c_3), \ldots$ for successive colour values may overlap in general, i.e., in general $T(c_i) \cap T(c_j) \neq \emptyset$ for some $i \neq j$ where $i \geq 1, j \geq 1$. For example, some vertices may already be assigned c_2 before all the vertices that will eventually be assigned c_1 have been assigned a colour. However, to obtain an upper bound on the running time of Algorithm 1, we consider the worst case where these processes do not overlap, and colours are assigned one by one, i.e., $T(c_i) \cap T(c_j) = \emptyset$ for every $i \neq j$ where $i \geq 1, j \geq 1$. In this worst case, by linearity of expectation, the expected time to compute the entire colouring is the sum of the expected times to compute $MIS(c_i)$, for $i = 1$ to at most $\Delta + 1$.

An analysis of the expected running time of a very similar process to compute a maximal independent set with the probability values chosen in the way specified in Algorithm 1 is given in [AABJ$^+$11] (see also Theorem 4.2.5). We say that an edge becomes *inactive* if either of its vertices is chosen as a member of the independent set. The analysis given in [AABJ$^+$11] shows that in every phase with more than $\log n$ time steps, at least some constant fraction of the edges of the graph are expected to become inactive. Hence the expected number of phases with more than $\log n$ time steps needed to complete the computation of $MIS(c)$ is $O(\log n)$.

To evaluate this sum, we note that the length of each successive phase increases by one. Hence, if we take $A \log n$ phases to complete process $MIS(c_1)$, for some constant A, where each phase consists of more than $\log n$ time steps, then the last of these phases consists of $(A + 1) \log n$ time

steps. After a further $A \log n$ phases the length of each phase will rise to $(2A+1)\log n$ time steps, and so on.

Hence the expected total number of time steps required by Algorithm 1 to compute a greedy colouring of G is at most the sum of the expected times to compute $MIS(c_i)$ for $i = 1$ to $\Delta + 1$, which is bounded by

$$\begin{aligned}
&A\log n \times (A+1)\log n + A\log n \times (2A+1)\log n + \ldots + A\log n \times ((\Delta+1)A+1)\log n \\
&= \sum_{i=1}^{\Delta+1} A(iA+1)\log^2 n \\
&= A(\Delta+1)(1+\frac{A}{2}(\Delta+2))\log^2 n,
\end{aligned}$$

which is $O(\Delta^2 \log^2 n)$. □

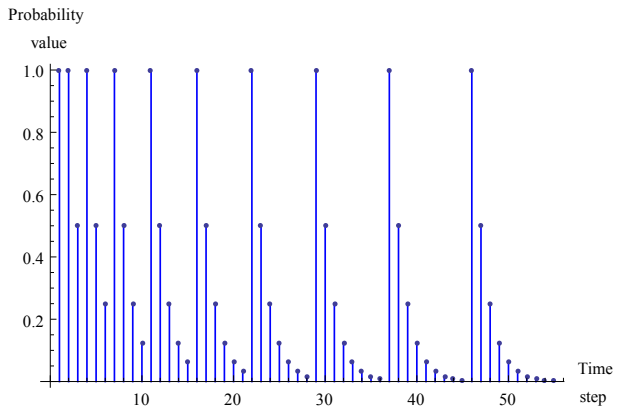
We have shown in Theorem 5.3.1 that the expected total number of time steps required by Algorithm 1 to compute a greedy colouring of a graph G with n vertices and maximum degree Δ is $O(\Delta^2 \log^2 n)$. Another important measure in distributed computing is the total number of messages sent (per processor), which is referred to as the message complexity. We will now establish an upper bound on the expected number of messages sent by each processor in a network of processors running Algorithm 1.

Theorem 5.3.2. *Given any graph G with n vertices and maximum degree Δ, the expected total number of messages sent by each processor running Algorithm 1 on G is $O(\Delta \log n)$.*

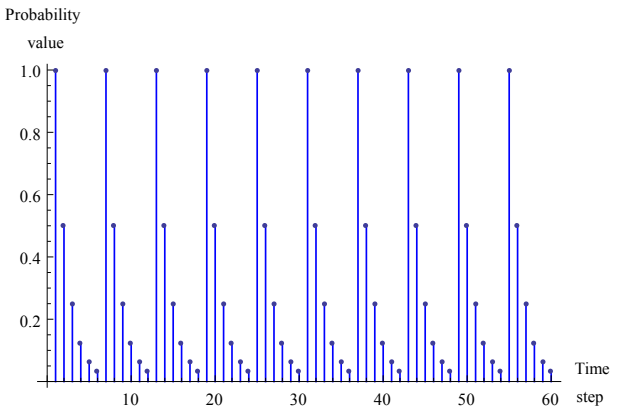
Proof. Let v be a vertex executing Algorithm 1, and consider the whole sequence of time steps until v becomes inactive, i.e., terminates its computation. We denote the probability that v sends a message at each step by p_t. The values of p_t in successive time steps will be as follows:
$1, 1, \frac{1}{2}, 1, \frac{1}{2}, \frac{1}{4}, 1, \frac{1}{2}, \frac{1}{4}, \frac{1}{8}, 1, \frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{16}, \ldots$ (see Figure 5.1a).

The expected number of messages sent during phase x is $\sum_{i=0}^{x-1} \frac{1}{2^i} < 2$.

As in the proof of Theorem 5.3.1 we assume for a worst-case analysis that the steps of the computation are partitioned into successive processes $MIS(c_1), MIS(c_2), MIS(c_3), \ldots$ for successive colour values $c_1, c_2, c_3, \ldots$ each of which computes a maximal independent set. By the analysis in [AABJ$^+$11], each of these processes is completed in $O(\log n)$ phases, and there are at most $\Delta + 1$ such processes, so the overall expected number of messages sent is $O(\Delta \log n)$. □



(a) Probability values chosen in Algorithm 1



(b) Probability values chosen in Algorithm 2

Figure 5.1: Comparison between probability values chosen in Algorithm 1 and Algorithm 2

5.3.2 Algorithm 2: distributed GC with only knowledge of n

Our second algorithm for distributed greedy colouring, Algorithm 2, is described in Table 5.3. As in Algorithm 1, we divide the computation into phases, but now each phase consists of a fixed number $\lceil \log n \rceil + 1$ of time steps. Hence to set the probability values at each processor we require some knowledge of the graph, namely, the number, n, of vertices.

An example of the probability values chosen by Algorithm 2 in the first 10 phases is shown in Figure 5.1b, where we have assumed that $4 < \log n \leq 5$, i.e., $\lceil \log n \rceil$=5, so each phase consists of exactly 6 time steps.

Table 5.3: Algorithm 2 at each processor

Global variable: n: the total number of vertices in the graph.
Local variables: i : time step counter;
p : local probability value;
TRYING : Boolean flag, initialised to FALSE;
S : Set of forbidden colours, i.e., those taken by neighbours, initialised to $\emptyset$.

1. Assume each processor knows the total number n of vertices in the graph;
2. Set $S \leftarrow \emptyset$;
3. **while** active, at each time step **do**
4. Set $i \leftarrow 0$;
5. **while** $i \leq \lceil \log n \rceil$ **do**
6. Set probability $p \leftarrow 1/2^i$;
7. Set $i \leftarrow i + 1$;
8. *FIRST EXCHANGE* (As in Table 5.1);
9. *SECOND EXCHANGE* (As in Table 5.1).

We analyse the performance of Algorithm 2 as follows:

Theorem 5.3.3. *Given any graph G with n vertices and maximum degree Δ, the expected total number of time steps required by Algorithm 2 to compute a greedy colouring of G is $O(\Delta \log^2 n)$.*

Proof. The fact that any complete computation of Algorithm 1 will correctly output a greedy colouring of G was established in Theorem 5.2.3.

By Lemma 5.2.4, we know that the vertices assigned colour value c constitute a maximal independent set in the subgraph induced by the vertices assigned colour value c or higher. We follow the poof of Theorem 5.3.1 by denoting the process that results in assigning colour c to some vertices to form this maximal independent set by $MIS(c)$, and denoting by $T(c)$ the set of time steps which attempt to assign colour c to some vertices to form $MIS(c)$.

In both Algorithm 1 and Algorithm 2, each vertex varies over time the probability that it attempts to choose a colour until it succeeds and becomes inactive. The only difference between Algorithm 1 and Algorithm 2 is that in Algorithm 2 each phase consists of a fixed number $\lceil \log n \rceil + 1$ of time steps, rather than a progressively larger number of time steps as in Algorithm 1.

As in the proof of Theorem 5.3.1, we note that the analysis given in [AABJ$^+$11] shows that in every phase with more than $\log n$ time steps, at least some constant fraction of the edges of the graph are expected to become inactive. Hence the expected number of phases with more than $\log n$ time steps needed for the computation of $MIS(c)$ is $O(\log n)$. However, in Algorithm 2, each phase consists of exactly $\lceil \log n \rceil + 1$ time steps, which is more than $\log n$ time steps.

We again consider the worst case where the processes $MIS(c_1), MIS(c_2), MIS(c_3), \ldots$ for successive colour values do not overlap, i.e., $T(c_i) \cap T(c_j) = \emptyset$ for every $i \neq j$ where $i \geq 1, j \geq 1$. By linearity of expectation, the expected time to compute the entire colouring is the sum of the expected times to compute $MIS(c_i)$, for $i = 1$ to at most $\Delta + 1$. If the expected number of phases to complete each process $MIS(c_i)$ is $B \log n$, for some constant B, then the expected total number of time steps required by Algorithm 2 is at most $(\Delta+1)B \log n(\lceil \log n \rceil + 1)$, which is $O(\Delta \log^2 n)$. □

We have shown that the expected total number of time steps required by Algorithm 2 to compute a greedy colouring of a graph G with n vertices and maximum degree Δ is $O(\Delta \log^2 n)$. We will now establish an upper bound on the expected number of messages sent by each processor in a network of processors running Algorithm 2.

Theorem 5.3.4. *Given any graph G with n vertices and maximum degree Δ, the expected total number of messages sent by each processor running Algorithm 2 on G is $O(\Delta \log n)$.*

Proof. Similar to the proof of Theorem 5.3.2. □

5.4 Summary

In this chapter, we have proposed and analysed a novel algorithmic scheme for the problem of distributed greedy colouring. This algorithmic scheme is derived from the approach used for the novel MIS selection algorithm inspired by the study of the neurological development of the fruit fly [AAB$^+$11, AABJ$^+$11]. Using this basic algorithmic scheme, we have constructed two efficient distributed algorithms for greedy graph colouring. Based on the restricted cellular computing model mentioned in Section 2.2.2, these new algorithms allow each processor to have minimal knowledge of the graph, and to send only simple messages corresponding to the set of colours to be used.

These two algorithms to distributed greedy colouring are extremely simple. However, they are not as efficient as previous state-of-the-art distributed greedy colouring algorithms (see Table 2.2). Thus, as in Section 4.3, the question still holds that whether the simple mechanism used by biological cells for "fine-grained" pattern formation has an inherently lower efficiency compared with carefully engineered algorithms. By looking more closely at the mechanism of "fine-grained" pattern formation during cell development, Chapter 6 will give an answer to this question.

CHAPTER 6

Improved distributed MIS selection and greedy colouring

In Chapter 4, we studied and implemented in P systems the novel bio-inspired distributed MIS selection algorthm that was proposed by Afek et al. in [AAB$^+$11, AABJ$^+$11]. In Chapter 5, we extended the novel bio-inspired distributed MIS selection algorithm to solve the distributed greedy colouring problem. These novel algorithms are simple and usable, however, not as efficient as previous state-of-the-art algorithms (see Theorem 4.3.1).

As mentioned in Chapter 2, the most well-known distributed algorithm for maximal independent set selection is the elegant randomized algorithm of [Lyn96, Wat07], generally known as Luby's algorithm, which has an expected running time which is $O(\log n)$. The randomised distributed MIS selection algorithm proposed in [MRSDZ11] also achieves a time complexity of $O(\log n)$ which is shown to be the optimal time complexity that can be achieved using one-bit messages [MRSDZ11] (see Table 2.1). With respect to distributed greedy colouring, Métivier et al. proposed a simple randomised distributed $(\Delta + 1)$-colouring algorithm requiring $O(\Delta + \log n)$ time [MRSDZ10]. This algorithm serves as the best known distributed greedy colouring algorithm in the literature (see Table 2.2).

Does this mean that the simple mechanism used by biological cells for "fine-grained" pattern formation has an inherently lower efficiency compared with carefully engineered algorithms such as Luby's algorithm? To answer that question we look more closely at the mechanism of "fine-grained" pattern formation during cell development: It is clear that the cells participating in this developmental process do not act autonomously – they continuously adjust their behaviour based on the signals being made by the cells around them.

In this chapter, we build on the work of Afek et al. which was inspired by studying the development of a network of cells in the fruit fly [AAB$^+$11]. However we incorporate for the first time

another important feature of the biological system: varying the probability value used at each node based on local feedback from neighbouring nodes.

More specifically, we first propose a randomised distributed MIS selection algorithm that is able to operate under very restrictive conditions. Our approach of distributed computing assumes an identical anonymous processor at each node that has *no information about the network*. At each time step, each node can only *broadcast a single bit* to all its neighbours, or remain silent. Each node can detect whether one or more neighbours have broadcast, but cannot tell how many neighbours have broadcast, *or which ones*.

We prove that our algorithm is optimal in both time and bit complexity for such a cellular computing model, by showing that it runs in expected $O(\log n)$ time, where n is the number of nodes, and the expected number of messages sent by each node is bounded by a constant, regardless of the network.

We then extend the approach to obtain an algorithm for the distributed greedy colouring problem. This algorithm also runs under very restrictive conditions where the processors are anonymous and have no information about the network. For this problem we allow each node to broadcast only a single message to all neighbours at each time step representing a single desired colour value. Once again nodes can only detect whether at least one neighbour has broadcast a colour, and cannot tell how many neighbours have broadcast, or which ones.

Our algorithm for greedy colouring is remarkably simple and computes a greedy colouring in expected $O(\Delta + \log n)$ time, where n is the number of nodes and Δ is the maximum degree of the network. Once again the expected total number of messages sent by each processor is bounded by a constant. Our improved greedy colouring algorithm works under very restrictive conditions and matches the best known time complexity for obtaining a greedy colouring.

To obtain our results we introduce a new form of analysis to determine the time complexity. Nearly all previous analytical techniques in this area have relied on a general technique, originally devised by Luby [Lub86], which divides the computation into successive phases and shows that some fixed fraction of the network is expected to be eliminated in each phase, so that there are at most logarithmically many phases. Our algorithms do not have this property, and hence require a more flexible form of analysis, which we describe in detail below.

We organize this chapter as follows: We first propose our new distributed MIS selection algorithm and analyse its worst-case time and message complexity. Then, we propose our new distributed greedy colouring algorithm with its worst-case time and message complexity analysis. Finally, we give a short summary of this chapter.

6.1 Improved distributed MIS selection algorithm

Our new algorithm uses a similar basic scheme to the one in Table 4.1, but changes the way that the probability value p varies over time (see Table 6.1). Inspired by the positive feedback mechanisms that control cellular processes [Bra06, CMML96], we give each node an independently updated probability value. These probabilities are initialised to arbitrary values (above some strictly positive fixed threshold value, p_0). They are decreased whenever one or more neighbouring nodes signal that they wish to join the independent set, and are increased whenever no neighbouring node issues such a signal. We allow each increase or decrease to be by some arbitrary factor f, which may vary at each step, but is bounded by the global parameters f_1 and f_2 (with $1 < f_1 \leq f_2$).

Table 6.1: The improved algorithm for distributed MIS selection with local feedback

Global constants: p_0 : lower bound on initial probability value;
f_1, f_2 : lower and upper bounds on change factor for probability value.

Local variables: p : local probability value, initialised to some value in $[p_0, 1]$;
f : change factor for probability value, chosen arbitrarily in $[f_1, f_2]$;
TRYING : Boolean flag, initialised to FALSE.

1. **while** active, at each time step **do**
2. *FIRST EXCHANGE*
3. With probability p, set TRYING $\leftarrow$ TRUE and **send** signal to all neighbours;
4. **Receive** any signals sent by neighbours;
5. Set f to some value in the interval $[f_1, f_2]$;
6. **if** any signal was received **then**
7. TRYING $\leftarrow$ FALSE and $p \leftarrow p/f$ (decrease p)
8. **else**
9. $p \leftarrow \min\{fp, 1\}$ (increase p)
10. *SECOND EXCHANGE*
11. **if** TRYING **then**
12. **Send** signal to all neighbours;
13. Join the MIS and terminate (become inactive).
14. **Receive** any signals sent by neighbours;
15. **if** any signal was received **then**
16. Terminate (become inactive)

Our main result below shows that varying the probabilities in this way, using a simple local feedback mechanism, gives an algorithm whose expected time to compute a maximal independent set is $O(\log n)$ (see Corollary 6.1.4, below). We also show that the expected number of signals sent by each node is bounded by a constant (see Theorem 6.1.5, below).

Note that the algorithm in Table 6.1 consists of two successive message exchanges. We shall refer to each such pair of message exchanges as a *time step* of the algorithm. Hence each time step contains two consecutive rounds of message exchanges.

Before we analyse the performance of this algorithm we first recall Theorem 4.3.1 which shows that the use of a feedback mechanism to adjust the probability values, as described in lines 5-9 of Table 6.1, is crucial to achieving the efficiency.

6.1.1 Time complexity with locally chosen probability values and feedback

Recall the preliminaries in Section 5.1. In this section we analyse the running time of our new algorithm for distributed MIS selection described in Table 6.1, where the probability values at each node are locally varied at each time step based on feedback from neighbouring nodes.

It follows from the analysis in Chapter 4 that if this algorithm terminates (i.e., all nodes become inactive) then it correctly identifies an MIS. The only question is the number of time steps required.

Note that, unlike Luby's algorithm [ABI86, Lub86], it is not true that in every time step we can expect at least some constant fraction of the edges to be incident to nodes that become inactive at that time step. For example, in a complete graph nodes will only become inactive when exactly one node signals. If all nodes are initialised with the same probability value and with the same (fixed) increase and decrease factor f, then all nodes will always possess the same probability value p_t. Whenever more than one node signals, all nodes will decrease their probabilities by f; if no node signals, all nodes will increase their probabilities by f. The probability of exactly one node signalling is thus $np_t(1-p_t)^{n-1}$ at each time step t. Hence, for complete graphs, with high probability all nodes will remain active for any fixed constant number of time steps. It follows that we must carry out a more detailed analysis over a sequence of time steps whose length increases with n.

Theorem 6.1.1. *For any fixed values of $p_0 > 0$, and $1 < f_1 \leq f_2$, there is a constant K_0 such that the following holds: For any graph G with n vertices, and any $k \geq 1$, the algorithm defined in Table 6.1 terminates in at most $K_0(k+1)\log n$ time steps, with probability at least $1 - O(1/n^k)$.*

Before beginning the proof of Theorem 6.1.1, it will be useful to define some notation and record a few simple facts. We will frequently use the well-known inequality

$$(1-\delta) \leq \exp(-\delta). \tag{6.1}$$

We will also use the following inequality, which holds for any $\lambda > 0$ and any $\delta \in [0, 1-e^{-\lambda}]$ (it holds with equality at the ends of this interval, and so holds at all points in between, by convexity).

$$(1-\delta) \geq \exp(-\delta\lambda/(1-e^{-\lambda})). \tag{6.2}$$

Finally, we will also need the following Chernoff-type inequality: if X is a sum of Bernoulli random variables, with expected value $\mathbb{E}X = m$, then for every $\delta > 0$,

$$\mathbb{P}[X > m + \delta] \leq \exp(-\delta^2/(2m + 2\delta/3)).$$

In particular,

$$\mathbb{P}[X > 2m] \leq \exp(-m/3). \tag{6.3}$$

We refer to sending a signal in the first exchange of the algorithm in Table 6.1 as "beeping", and receiving such a signal from a neighbour as "hearing a beep".

For any vertex v, we define $\mu_t(v)$, which we call the "weight" of v, to be the probability that v beeps at time step t. (By convention, we set $\mu_t(v) = 0$ if v is inactive at time t; this simplifies notation, while allowing us to ignore the contribution of inactive vertices.) Note that μ_t is a random measure, as it depends on the beeps of other vertices during the first $t-1$ time steps. For any $W \subseteq V$ we write $\mu_t(W)$ for $\sum_{v \in W} \mu_t(v)$.

Recall that the set of vertices adjacent to a given vertex v is called the set of *neighbours* of v, and denoted by $\Gamma(v)$.

Definition 6.1.2. *For any $\lambda > 0$, a vertex v will be called* λ-light *at time step t if $\mu_t(\Gamma(v)) \leq \lambda$ and every neighbour of v has weight at most $1 - \exp(-\lambda)$; otherwise, vertex v is called* λ-heavy.

For any vertex that is λ-*light*, the weight of each of its neighbours individually is bounded by $1 - \exp(-\lambda)$ and the sum of all its neighbours' weights is not too large (and so the vertex is not too likely to hear a beep at time t). Note that a fixed vertex may move back and forth between being λ-heavy and λ-light over time.

Our first result establishes a lower bound on the probability that at least one vertex in a set of λ-light vertices will be added to the independent set at the current time step.

Lemma 6.1.3. *Let W be a set of vertices that are λ-light at time step t. The probability that at least one vertex in W is added to the independent set at time step t is at least $e^{-\phi\lambda}(1 - e^{-\mu_t(W)})$ where $\phi = \lambda/(1 - \exp(-\lambda))$.*

Proof. Let us order the vertices of W as $w_1, \ldots, w_m$, where $m = |W|$. The probability that some vertex of W is added to the independent set at time step t is at least the probability that the earliest vertex of W that beeps at time step t is added to the independent set. For $i = 1, \ldots, m$, define events E_i and F_i by

$$E_i = (w_i \text{ beeps; } w_1, \ldots, w_{i-1} \text{ do not beep})$$

$$F_i = (\text{no neighbour of } w_i \text{ beeps}).$$

The events $E_i \cap F_i$ are pairwise disjoint, so using the definition of conditional probability, we have that the probability that the earliest vertex of W that beeps is added to the independent set is

$$\mathbb{P}[\bigcup_{i=1}^{m}(E_i \cap F_i)] = \sum_{i=1}^{m} \mathbb{P}[E_i \cap F_i] = \sum_{i=1}^{m} \mathbb{P}[E_i]\mathbb{P}[F_i|E_i].$$

It is easily seen that $\mathbb{P}[F_i|E_i] \geq \mathbb{P}[F_i]$ since $\mathbb{P}[F_i|E_i]$ is conditioned on the event that w_i beeps and $w_1, \ldots, w_{i-1}$ do not. Hence we have

$$\mathbb{P}[F_i|E_i] \geq \mathbb{P}[F_i] = \prod_{v \in \Gamma(w_i)} (1 - \mu_t(v))$$

Since w_i is λ-light, we may apply Inequality (6.2), to conclude that

$$\prod_{v \in \Gamma(w_i)} (1 - \mu_t(v)) \geq \prod_{v \in \Gamma(w_i)} \exp(-\phi\mu_t(v)) = \exp(-\phi\mu_t(\Gamma(w_i))) \geq \exp(-\phi\lambda)$$

where $\phi = \lambda/(1 - \exp(-\lambda))$. Hence we have

$$\sum_{i=1}^{m} \mathbb{P}[E_i]\mathbb{P}[F_i|E_i] \geq \exp(-\phi\lambda) \sum_{i=1}^{m} \mathbb{P}[E_i].$$

But $\sum_{i=1}^{m} \mathbb{P}[E_i]$ is simply the probability that some vertex in W beeps, which is given by $1 - \prod_{v \in W}(1 - \mu_t(v))$. Using Inequality (6.1) this value is at least $1 - \exp(-\mu_t(W))$.

Thus the probability that some vertex of W is added to the independent set at time step t is at least

$$\exp(-\phi\lambda) \sum_{i=1}^{m} \mathbb{P}[E_i] \geq e^{-\phi\lambda}(1 - e^{-\mu_t(W)}),$$

where $\phi = \lambda/(1 - \exp(-\lambda))$. □

Proof of Theorem 6.1.1. Fix an arbitrary vertex v. We shall show that, with failure probability $O(1/n^{k+1})$, v becomes inactive within $K_0(k+1)\log n$ time steps, for a suitable choice of constant K_0. Taking a union bound over all n choices of v, it follows that with failure probability $O(1/n^k)$ every vertex becomes inactive and the algorithm terminates within $K_0(k+1)\log n$ time steps, which proves the theorem.

At each time step $t \geq 1$, we partition the neighbourhood of v into λ-light and λ-heavy vertices, for a suitable fixed choice of λ

$$L_t = L_t(v) = \{x \in \Gamma(v) \mid x \text{ is } \lambda\text{-light at step } t\}$$
$$H_t = H_t(v) = \{x \in \Gamma(v) \mid x \text{ is } \lambda\text{-heavy at step } t\}.$$

We will follow the behaviour of $\mu_t(L_t)$ and $\mu_t(H_t)$ over time.

The idea of the argument is roughly as follows: if $\mu_t(L_t)$ is large at many time steps, then by Lemma 6.1.3 it is very likely that some neighbour of v will be added to the independent set on one of these occasions, leading to v becoming inactive (see Claim 1). If this does not happen, then $\mu_t(L_t)$ must be small most of the time. Now consider H_t. Vertices that are λ-heavy at time t are likely to hear beeps and so drop in weight (as their signalling probability is reduced); it will follow that with high probability $\mu_{t+1}(H_t)$ is a constant factor smaller than $\mu_t(H_t)$ most of the time (see Claims 2 and 3). Now we look at the evolution of $\mu_t(\Gamma(v))$, the weight of the whole neighbourhood of v. It may be large and increasing for some small fraction of the time, but mostly it is either shrinking or else it is already small (see Claim 4). It will follow that, for at least some fixed fraction of the time, $\mu_t(\Gamma(v))$ is small. But this implies that, for at least some fixed fraction of the time, v will not hear any beeps, and hence $\mu_t(v)$ will be large for some fixed fraction of the time (see Claim 5). This implies that it is very likely that at some point in the sequence of time steps we are considering v will beep and not hear any beeps, and so get added to the independent set (see Claim 6).

To make this argument precise, we now define the following constants:

$$\begin{aligned}
r &= 1 + (\log f_2 / \log f_1);\\
\lambda &= \log(32r(r+2)(f_2 - f_1^{-1})/(f_2^{-1/r} - f_1^{-1}));\\
\phi &= \lambda/(1-\exp(-\lambda));\\
\beta &= 1/(4(r+2)\phi f_2);\\
\alpha &= (\beta/2)(f_2^{-1/r} - f_1^{-1})/(f_2 - f_1^{-1});\\
K_0 &= (8r(r+2))\max\{6, 1/p_0, 1/\log f_2, 1/(e^{-\phi\lambda}(1-e^{-\alpha}))\};
\end{aligned}$$

The values of these constants depend only on the fixed parameters f_1 and f_2 which bound the probability update factor f used in the algorithm, and on the initial minimum probability threshold p_0 (see Table 6.1). Note that $1 < f_1 \leq f_2$, so $r \geq 2$ and $\lambda > \log 256 > 5$.

To simplify the presentation, we also define

$$K = K_0(k+1).$$

At each time step t where v is active, we consider the following four possible events:

(E1) $\mu_t(L_t) \geq \alpha$ ['$\Gamma(v)$ has a significant weight of light neighbours']

(E2) $\mu_t(L_t) < \alpha$ and $\mu_t(\Gamma(v)) \leq \beta$ ['the neighbourhood of v is very light']

(E3) $\mu_t(L_t) < \alpha$, $\mu_t(\Gamma(v)) > \beta$ and $\mu_{t+1}(\Gamma(v)) \leq f_2^{-1/r}\mu_t(\Gamma(v))$ ['the neighbourhood of v shrinks significantly in weight during time step t']

(E4) $\mu_t(L_t) < \alpha$, $\mu_t(\Gamma(v)) > \beta$ and $\mu_{t+1}(\Gamma(v)) > f_2^{-1/r}\mu_t(\Gamma(v))$ ['the neighbourhood of v does not shrink significantly in weight during time step t (and may grow)']

Exactly one of these events must occur at each time step where v is active.

We organize the rest of the proof as a series of claims.

Claim 1. *With failure probability $O(1/n^{k+1})$, (E1) occurs at most $(K \log n)/(8r(r+2))$ times in the first $K \log n$ time steps.*

Each time that (E1) occurs, it follows from Lemma 6.1.3 that with probability at least $e^{-\phi\lambda}(1 - e^{-\mu_t(L_t)}) \geq e^{-\phi\lambda}(1 - e^{-\alpha})$ some vertex of L_t is added to the independent set (and so v becomes inactive and the process at v terminates). Let $\phi_1 = e^{-\phi\lambda}(1 - e^{-\alpha})$: the probability that (E1) occurs $(K \log n)/(8r(r+2))$ times without v becoming inactive is at most $(1 - \phi_1)^{(K \log n)/(8r(r+2))} \leq \exp(-(\phi_1 K_0/(8r(r+2)))(k+1)\log n)$. By our choice of K_0, we have $K_0 \geq (8r(r+2))/\phi_1$, so this probability is at most $\exp(-(k+1)\log n) = n^{-(k+1)}$. This proves Claim 1.

The bad event for us will be (E4), so let us bound the probability that (E4) occurs.

Claim 2. *At each time step t, the probability that (E4) occurs is at most $1/(16r(r+2))$.*

If (E4) can occur, then we must have $\mu_t(L_t) < \alpha$ and $\mu_t(\Gamma(v)) > \beta$. For any $x \in H_t$, there are two cases to consider - the first is that the total weight of all its neighbouring vertices is greater than λ; the second is that at least one of its neighbouring vertices individually has weight more than $1 - \exp(-\lambda)$.

In the first case, using Inequality (6.1), the probability that no neighbour of x beeps at time step t is at most $\exp(-\mu_t(\Gamma(x))) \leq \exp(-\lambda)$. In the second case, the probability that no neighbour of x beeps at time step t is still at most $\exp(-\lambda)$. Thus, for any $x \in H_t$ we have shown that the probability that no neighbour of x beeps at time step t is at most $\exp(-\lambda)$.

Let H_t^0 be the set of vertices in H_t that do not hear a beep at time step t, and let $H_t^1 = H_t \setminus H_t^0$ be the remaining vertices in H_t that do hear a beep. Then $\mathbb{E}[\mu_t(H_t^0)] \leq \exp(-\lambda)\mu_t(H_t)$, and so by Markov's inequality

$$\mathbb{P}[\mu_t(H_t^0) \geq 16r(r+2)\exp(-\lambda)\mu_t(H_t)] \leq 1/(16r(r+2)). \tag{6.4}$$

Now all vertices in H_t^1 decrease their weight by a factor of at least f_1, while vertices in L_t and H_t^0 may either decrease or increase their weight (additionally, some weights may get set to 0 if vertices become inactive). So

$$\begin{aligned}\mu_{t+1}(\Gamma(v)) &\leq \frac{1}{f_1}\mu_t(H_t^1) + f_2\mu_t(H_t^0) + f_2\mu_t(L_t) \\ &= \frac{1}{f_1}\mu_t(\Gamma(v)) + (f_2 - \frac{1}{f_1})\mu_t(H_t^0) + (f_2 - \frac{1}{f_1})\mu_t(L_t)\end{aligned}$$

It follows from (6.4) that, with probability at least $1 - 1/(16r(r+2))$,

$$\begin{aligned}\mu_{t+1}(\Gamma(v)) &\leq \frac{1}{f_1}\mu_t(\Gamma(v)) + (f_2 - \frac{1}{f_1})16r(r+2)e^{-\lambda}\mu_t(H_t) + (f_2 - \frac{1}{f_1})\mu_t(L_t) \\ &\leq f_2^{-1/r}\mu_t(\Gamma(v)),\end{aligned}$$

where the final inequality follows from our choice of λ, which gives

$$(f_2 - f_1^{-1})16r(r+2)e^{-\lambda}\mu_t(H_t) \leq (f_2 - f_1^{-1})16r(r+2)e^{-\lambda}\mu_t(\Gamma(v)) \leq 1/2(f_2^{-1/r} - f_1^{-1})\mu_t(\Gamma(v)),$$

and our choice of α and β, because we are assuming that $\mu_t(L_t) < \alpha$ and $\mu_t(\Gamma(v)) > \beta$, so we have

$$(f_2 - f_1^{-1})\mu_t(L_t) < (f_2 - f_1^{-1})\alpha = 1/2(f_2^{-1/r} - f_1^{-1})\beta < 1/2(f_2^{-1/r} - f_1^{-1})\mu_t(\Gamma(v)).$$

Thus the probability that (E4) occurs is at most $1/(16r(r+2))$. This proves Claim 2.

Claim 3. *With failure probability $O(1/n^{k+1})$, (E4) occurs at most $(K\log n)/(8r(r+2))$ times in the first $K\log n$ time steps.*

At each time step, the probability of (E4) depends on the past history of the process. However, by Claim 2, it is always at most $1/(16r(r+2))$, and so we can couple occurrences of (E4) with a sequence of independent events each occurring with probability $1/(16r(r+2))$. It follows that the number of occurrences of (E4) in the first $K\log n$ time steps is stochastically dominated by a binomial random variable X with parameters $K\log n$ and $1/(16r(r+2))$. The probability that (E4) occurs more than $(K\log n)/(8r(r+2))$ times is therefore, by (6.3), at most

$$\mathbb{P}[X > 2\mathbb{E}X] \leq \exp(-\mathbb{E}X/3) \leq \exp(-(K\log n)/(48r(r+2))).$$

By our choice of K_0, we have $K_0 \geq 48r(r+2)$, so this probability is $O(n^{-(k+1)})$, which proves Claim 3.

From Claim 1 and Claim 3, we conclude that with failure probability $O(n^{-(k+1)})$, (E1) and (E4) altogether occur at most $K(\log n)/(4r(r+2))$ times in the first $K\log n$ time steps. We next show that, with small failure probability, $\mu_t(\Gamma(v))$ is small most of the time.

Claim 4. *With failure probability $O(1/n^{k+1})$, $\mu_t(\Gamma(v)) > f_2\beta$ at most $(K\log n)/(2(r+2))$ times in the first $K\log n$ time steps.*

Let T be the set of time steps $t \geq 1$ at which $\mu_t(\Gamma(v)) > f_2\beta$. We decompose T into (maximal) intervals of integers, say as $T_1 \cup \cdots \cup T_m$. Let $T_i = [s_i, t_i]$ be one of these intervals. We colour each integer $t \in T_i$ *red* if (E1) or (E4) occurred at the previous time step, and *blue* if (E3) occurred (note that (E2) cannot occur, as $\mu_{t-1}(\Gamma(v)) \geq \mu_t(\Gamma(v))/f_2 > \beta$). By the definition of (E3), we have

$\mu_t(\Gamma(v)) \leq f_2^{-1/r}\mu_{t-1}(\Gamma(v))$ at blue time steps, and we have $\mu_t(\Gamma(v)) \leq f_2\mu_{t-1}(\Gamma(v))$ otherwise. Let r_i be the number of red elements in T_i and b_i the number of blue elements. It follows that

$$\mu_{t_i}(\Gamma(v)) \leq \mu_{s_i-1}(\Gamma(v)) \cdot {f_2}^{r_i - b_i/r}.$$

Since $\mu_{t_i}(\Gamma(v)) > f_2\beta$ it follows that

$$r_i > \frac{1}{r}b_i + \log_{f_2} f_2\beta - \log_{f_2}(\mu_{s_i-1}(\Gamma(v))).$$

However, $\mu_{s_i-1}(\Gamma(v)) \leq f_2\beta$ in all cases where $s_i > 1$, and $\mu_0(\Gamma(v)) < n$. Summing over i, we see that

$$\sum_{i=1}^{m} r_i > \frac{1}{r}\sum_{i=1}^{m} b_i + \log_{f_2} f_2\beta - \log_{f_2} n$$

But red time steps correspond to events (E1) and (E4), and we have already shown that these occur at most $(K\log n)/(4r(r+2))$ times altogether in the first $K\log n$ time steps. Hence the total number of time steps in T, both red and blue, is less than $(K\log n)/(4r(r+2)) + (K\log n)/(4(r+2)) + r\log n/\log f_2$. By our choice of K_0, this is less than $(K\log n)/(2(r+2))$. This proves Claim 4.

Claim 5. *With failure probability $O(1/n^{k+1})$, v hears a beep at most $(K\log n)/(r+2)$ times in the first $K\log n$ time steps.*

By our choice of β, we have that $f_2\beta = (1-e^{-\lambda})/(4(r+2)\lambda) < (1-e^{-\lambda})$. Hence we may apply Inequality (6.2), to show that when $\mu_t(\Gamma(v)) \leq f_2\beta$ the probability that v hears no beep is

$$\prod_{x\in\Gamma(v)}(1-\mu_t(x)) \geq \prod_{x\in\Gamma(v)} \exp(-\phi\mu_t(x)) \geq \exp(-\phi\mu_t(\Gamma(v))) \geq \exp(-\phi f_2\beta) \geq 1-\phi f_2\beta,$$

and so v hears a beep with probability at most $\phi f_2\beta = 1/(4(r+2))$. Using Inequality (6.3) this implies that with failure probability $O(n^{-(k+1)})$ there are at most $(K\log n)/(2(r+2))$ time steps among the first $K\log n$ at which $\mu_t(\Gamma(v)) \leq f_2\beta$ and v hears a beep. By Claim 4, with the same failure probability, there are also at most $(K\log n)/(2(r+2))$ time steps at which $\mu_t(\Gamma(v)) > f_2\beta$ (and v might hear a beep at any of these steps). It follows that, with failure probability $O(n^{-(k+1)})$, v hears a beep at most $(K\log n)/(r+2)$ times in the first $K\log n$ time steps. This proves Claim 5.

Claim 6. *With failure probability $O(1/n^{k+1})$, v becomes inactive during the first $K\log n$ time steps.*

From the previous claim, we may assume that v hears a beep on at most $K\log n/(r+2)$ occasions during the first $K\log n$ time steps. On these occasions it decreases its local probability value p by a factor of at most f_2. We shall refer to these as red steps.

Hence there are at least $\frac{(r+1)}{(r+2)}K\log n$ time steps during the first $K\log n$ time steps where v does not hear a beep, so it either terminates, or increases its local probability value p by a factor of at least f_1, or else increases p to 1. We shall refer to these as blue time steps. Note that if v beeps in a blue

step then it will terminate in that time step. Hence a blue time step where the value of p increases to 1 must be immediately followed by a red time step, or a blue time step where v terminates.

Now, by our choice of r, $f_1^r > f_2$. This means that there must be at least $\frac{1}{(r+2)}K \log n$ blue time steps during the first $K \log n$ time steps where either v has terminated, or else the local probability value p at v is at least as high as the initial value, p_0.

The probability that v will terminate at each of these blue time steps is at least p_0, so the probability that v remains active throughout all these blue time steps is at most $(1-p_0)^{K \log n/(r+2)}$. Using Inequality (6.1), this means that the probability that v remains active throughout these time steps is at most $\exp(-p_0 K \log n/(r+2))$. By our choice of K_0, we have $K_0 > (r+2)/p_0$, so this is $O(n^{-(k+1)})$. Hence v terminates with failure probability $O(n^{-(k+1)})$.

This proves Claim 6, and completes the proof of Theorem 6.1.1. □

Corollary 6.1.4. *The expected number of time steps taken by the algorithm in Table 6.1 on any graph with n nodes is $O(\log n)$.*

Proof. Let T be the total number of time steps taken by the algorithm and let $T' = \lceil T/(K_0 \log n) \rceil$, where K_0 is the constant identified in Theorem 6.1.1.

By Theorem 6.1.1, we have that, for any $k \geq 1$, $\mathbb{P}[T' > k+1] \leq c'/n^k$ for some constant c'. Hence $\mathbb{E}[T'] = \sum_{k \geq 1} \mathbb{P}[T' \geq k] = O(1)$. □

6.1.2 Expected number of signals

In this section we will show that the expected number of times that each node signals is bounded by a constant. Hence the expected bit complexity per node for this algorithm *does not increase at all* with the number of nodes.

Theorem 6.1.5. *The expected total number of signals broadcast by any node executing the algorithm in Table 6.1 is $O(1)$.*

Proof. Let v be a node executing the algorithm in Table 6.1, and consider the whole sequence of time steps until v becomes inactive.

During each time step, one of the following 3 things happens:

Case 1 v hears a beep from its neighbours, and so decreases its probability of beeping by a factor of f (from p_t to $\frac{1}{f}p_t$). We will call these *red* time steps.

Case 2 v hears no beep from its neighbours, and so increases its probability of beeping by a factor of f (from p_t to fp_t). We will call these *blue* time steps.

Case 3 v hears no beep from its neighbours, and so increases its probability of beeping to 1. We will call these *dark blue* time steps.

If v beeps in any blue or dark blue time step then it joins the MIS and becomes inactive, so the total number of beeps at such time steps is at most one, at the final time step in the sequence. Hence we only need to consider the expected number of beeps at red time steps.

Consider first those red time steps (if any) where the value of p_t is at its lowest point in the sequence so far. The expected number of times that v beeps during this subsequence of time steps is bounded by $p_1 + \frac{p_1}{f_1} + \frac{p_1}{f_1^2} + \frac{p_1}{f_1^3} \cdots \leq \frac{f_1}{f_1 - 1} p_1 \leq \frac{f_1}{f_1 - 1}$.

At all of the remaining red time steps the value of p_t is not at its lowest point so far, so it was lower at some previous blue step. Hence each of these red steps can be associated with a corresponding earlier blue step: the most recent blue step where the value of p_t was lower. We now define the constant $r = \lceil \log f_2 / \log f_1 \rceil$, where f_1 and f_2 denote respectively the lower bound and upper bound of f as given in Table 6.1. Note that $f_1^r \geq f_2$. Since blue time steps increase the value of p_t by at most a factor of f_2, and red time steps decrease the value of p_t by at least a factor of f_1, it follows that each blue time step will be associated with at most r red time steps.

Hence we have partitioned the remaining red time steps into groups of at most r, each associated with a single (earlier) blue step. To count the number of beeps we consider only those groups where v beeps at one or more of the time steps in the group. The expected number of beeps on red time steps in such groups is at most r times the expected number of events where v beeps at a red time step in the group but fails to beep at the associated blue time step. Since the value of p_t at a red time step is at most f_2 times as large as the value at the associated blue time step, the probability of beeping in the red time step and not at the associated blue time step is at most $f_2/(f_2 + 1)$. This means that the probability of terminating in each of the groups we are considering is at least $1/(f_2 + 1)$. Hence the expected number of beeps added in these groups before terminating is at most $r(f_2 + 1)$.

We have shown that the expected number of times that v beeps is at most
$1 + \frac{f_1}{f_1 - 1} + (\lceil \log f_2 / \log f_1 \rceil)(f_2 + 1)$, which proves the result. □

6.2 Improved distributed greedy colouring algorithm

Our new algorithm for distributed greedy colouring (see Table 6.2) is similar to our new distributed MIS selection algorithm. At each time step, each node may choose, with a certain probability p, to broadcast its first available colour to all its neighbours, indicating that it wishes to use that colour. If two neighbouring nodes broadcast the same colour at the same time step, then they will both abandon choosing that colour at that time step. On the other hand, if a node broadcasts a colour and none of its neighbouring nodes broadcast the same colour at that time step, then it will be successfully coloured, and will notify all its neighbouring nodes that they are forbidden to use that colour.

Inspired by the positive feedback mechanisms that control cellular processes, the probability value p is independently updated at each node at each time step using feedback from neighbouring

Table 6.2: The improved algorithm for distributed greedy colouring with local feedback

Global constants: p_0 : lower bound on initial probability value;
f_1, f_2 : lower and upper bounds on change factor for probability value.

Local variables: p : local probability value, initialised to some value in $[p_0, 1]$;
f : change factor for probability value, chosen arbitrarily in $[f_1, f_2]$;
TRYING : Boolean flag, initialised to FALSE;
S : Set of forbidden colours, i.e., those taken by neighbours, initialised to $\emptyset$.

1. **while** active, at each time step **do**
2. *FIRST EXCHANGE*
3. Choose the smallest available colour c that is not in S;
4. With probability p, set TRYING $\leftarrow$ TRUE and **send** c to all neighbours;
5. **Receive** any colour signals sent by neighbours;
6. Set f to some value in the interval $[f_1, f_2]$;
7. **if** any neighbour sent colour c **then**
8. TRYING $\leftarrow$ FALSE and $p \leftarrow p/f$ (decrease p)
9. **else**
10. $p \leftarrow \min\{fp, 1\}$ (increase p)
11. *SECOND EXCHANGE*
12. **if** TRYING **then**
13. **Send** c to all neighbours;
14. Assign colour c to this node and terminate (become inactive).
15. **Receive** any colour signals sent by neighbours and add them to S.

nodes. The value of p is initialised to some arbitrary value (above some strictly positive fixed threshold value, p_0). It is decreased whenever one or more neighbouring nodes broadcast the same colour, and is increased whenever no neighbouring node broadcasts the same colour. As in our MIS selection algorithm, we allow each increase or decrease to be by some arbitrary factor f, which may vary at each time step, but is always bounded by the global parameters f_1 and f_2 (with $1 < f_1 \leq f_2$).

The correctness of the algorithm in Table 6.2 follows easily from the two facts below (see also Section 5.2):

Fact 1 No two nodes that are assigned the same colour at the same time step are adjacent in the graph (see Lemma 5.2.1).

Fact 2 The colour assigned to any node is the smallest colour that is different from all colours previously assigned to neighbouring nodes (see Lemma 5.2.2).

Thus, if the algorithm in Table 6.2 is run on the nodes of any graph G, and all nodes become inactive, then the colour assigned to each of the nodes defines a greedy colouring of G.

Our analysis of the distributed greedy colouring algorithm shown in Table 6.2 is very similar to the analysis for the MIS selection algorithm given in Section 6.1.1.

Theorem 6.2.1. *For any fixed values of $p_0 > 0$, and $1 < f_1 \leq f_2$, there is a constant K_0 and a constant r such that the following holds: For any graph G with n vertices and maximum degree Δ, and any $k \geq 1$, the algorithm defined in Table 6.2 terminates in at most $8r(r+2)\Delta + K_0(k+1)\log n$ time steps, with probability at least $1 - O(1/n^k)$.*

As in Section 6.1.1, we refer to broadcasting any colour c in the first exchange as "beeping", and receiving *the same* colour c from a neighbour in that exchange as "hearing a beep". As before, for any vertex v, at any time step t, we define the measure $\mu_t(v)$, called the "weight" of v, to be the probability that v beeps at time step t.

The set of neighbours of a vertex v which are competing for the same colour as v at any time step will be called the *homogeneous* neighbours of v, and will be denoted by $\Gamma^{(h)}(v)$. We adapt Definition 6.1.2 to refer to homogeneous neighbours only, as follows.

Definition 6.2.2. *For any $\lambda > 0$, a vertex v will be called λ-light$^{(h)}$ at time step t if $\mu_t(\Gamma^{(h)}(v)) \leq \lambda$ and every homogeneous neighbour of v has weight at most $1 - \exp(-\lambda)$; otherwise, vertex v is called λ-heavy$^{(h)}$.*

As in Section 6.1.1, we first establish a lower bound on the probability that at least one vertex in a set of λ-light$^{(h)}$ vertices will be coloured at each time step.

Lemma 6.2.3. *Let W be a set of vertices that are λ-light$^{(h)}$ at time step t. The probability that at least one vertex in W gets coloured at time step t is at least $e^{-\phi\lambda}(1 - e^{-\mu_t(W)})$ where $\phi = \lambda/(1 - \exp(-\lambda))$.*

Proof. Identical to the proof of Lemma 6.1.3. □

Proof of Theorem 6.2.1. Fix an arbitrary vertex v. We use essentially the same argument as in the proof of Theorem 6.1.1, partitioning the neighbourhood of v into λ-light$^{(h)}$ and λ-heavy$^{(h)}$ vertices, and following the progress of these sets over time. Note that we consider the entire neighbourhood of v, not just the homogeneous neighbours. We show that the weight of this entire neighbourhood is small for at least a fixed fraction of the time, and hence v fails to receive any colour signals for at least a fixed fraction of the time.

We define the same constants, and the same events $(E1) - (E4)$ (see page 14). However, in this proof we will need to allow for the possibility that one or more neighbours of v are successfully coloured at any time step, which can happen up to Δ times, and does not immediately force v to become inactive. This means that we obtain a weaker upper bound on the number of occurrences of $(E1)$, and hence need to consider a longer sequence of time steps overall.

Following the same argument as in the proof of Claim 1, but allowing for up to Δ time steps where some neighbour of v is coloured, we obtain that with failure probability $O(1/n^{k+1})$, (E1) occurs at most $\Delta + (K \log n)/(8r(r+2))$ times in the first $8r(r+2)\Delta + K \log n$ time steps.

Following exactly the same arguments as in the proof of Claims 2 and 3, we obtain that with failure probability $O(1/n^{k+1})$, (E4) occurs at most $\Delta + (K \log n)/(8r(r+2))$ times in the first $8r(r+2)\Delta + K \log n$ time steps.

Next, following the same argument as in the proof of Claim 4, but using the weaker bound of $2\Delta + (K \log n)/(4r(r+2))$ for the total number of red time steps, we obtain that with failure probability $O(1/n^{k+1})$, $\mu_t(\Gamma(v)) > f_2\beta$ at most $4r\Delta + (K \log n)/(2(r+2))$ times in the first $8r(r+2)\Delta + K \log n$ time steps.

Now, using the same argument as in Claim 5 we can show that with failure probability $O(1/n^{k+1})$, the fraction of time steps where v hears a beep (or in fact receives any colour signal in the first exchange) during the first $8r(r+2)\Delta + K \log n$ time steps is at most $1/(r+2)$.

Finally, using the same argument as in Claim 6, this implies that with failure probability $O(1/n^{k+1})$, v becomes inactive during the first $8r(r+2)\Delta + K \log n$ time steps.

Taking a union bound over all possible choices of v, as before, gives the result. □

Corollary 6.2.4. *The expected number of time steps taken by the algorithm in Table 6.2 on any graph with n nodes is $O(\Delta + \log n)$.*

Finally, the proof of Theorem 6.1.5 considers only an individual node and does not take into account whether the neighbours of a node become inactive or not, so this proof applies equally well to our distributed colouring algorithm, giving the following result.

Theorem 6.2.5. *The expected total number of beeps broadcast by any node executing the algorithm in Table 6.2 is $O(1)$.*

Proof. Similar to the proof of Theorem 6.1.5. □

6.3 Summary

In this chapter, we have proposed an improved distributed MIS selection algorithm and an improved distributed greedy colouring algorithm, by incorporating another important feature of biological systems: varying the probability value used at each node based on local feedback from neighbouring nodes. Both our improved algorithms are simple and robust and also achieve the state-of-the-art performance in terms of their worst-case time complexity and message complexity. Moreover, the technique used to analyse these algorithms is novel and can be used when traditional analytical techniques fail.

Thus, by looking more closely at the mechanism of "fine-grained" pattern formation during cell development, we have answered the question left in Chapter 4 and Chapter 5 by showing that the simple mechanism inspired by biological cells for "fine grained" pattern formation has comparable efficiency with carefully engineered algorithms and is also simpler and more robust.

Even though we have analytically proved the worst case time and message complexity for our proposed algorithms in this chapter and previous chapters, their practical performance is still unknown. Thus, it is necessary to carry out some experiments to compare their practical running performance. Systematical experimental results illustrating the time complexity and message complexity of our proposed algorithms in practice are given in Chapter 7.

CHAPTER 7

Experimental results

We have studied different distributed MIS selection and greedy colouring algorithms and investigated their time and message complexity. However, the worst-case asymptotic upper bounds do not tell us the full story about how well the algorithms will perform in practice. In this chapter, we systematically examine the practical performance of the distributed MIS selection and greedy colouring algorithms.

We first investigate two versions of distributed MIS selection algorithms: the algorithm proposed by Afek et al. in [AABJ+11] in Chapter 4 (Table 4.1) and our improved MIS selection algorithm described in Chapter 6 (Table 6.1). We compare their practical time and message complexity on some standard families of graphs. In addition, we also investigate the size of MIS selected on some standard maximum clique and maximum independent set (MaxIS) benchmarks.

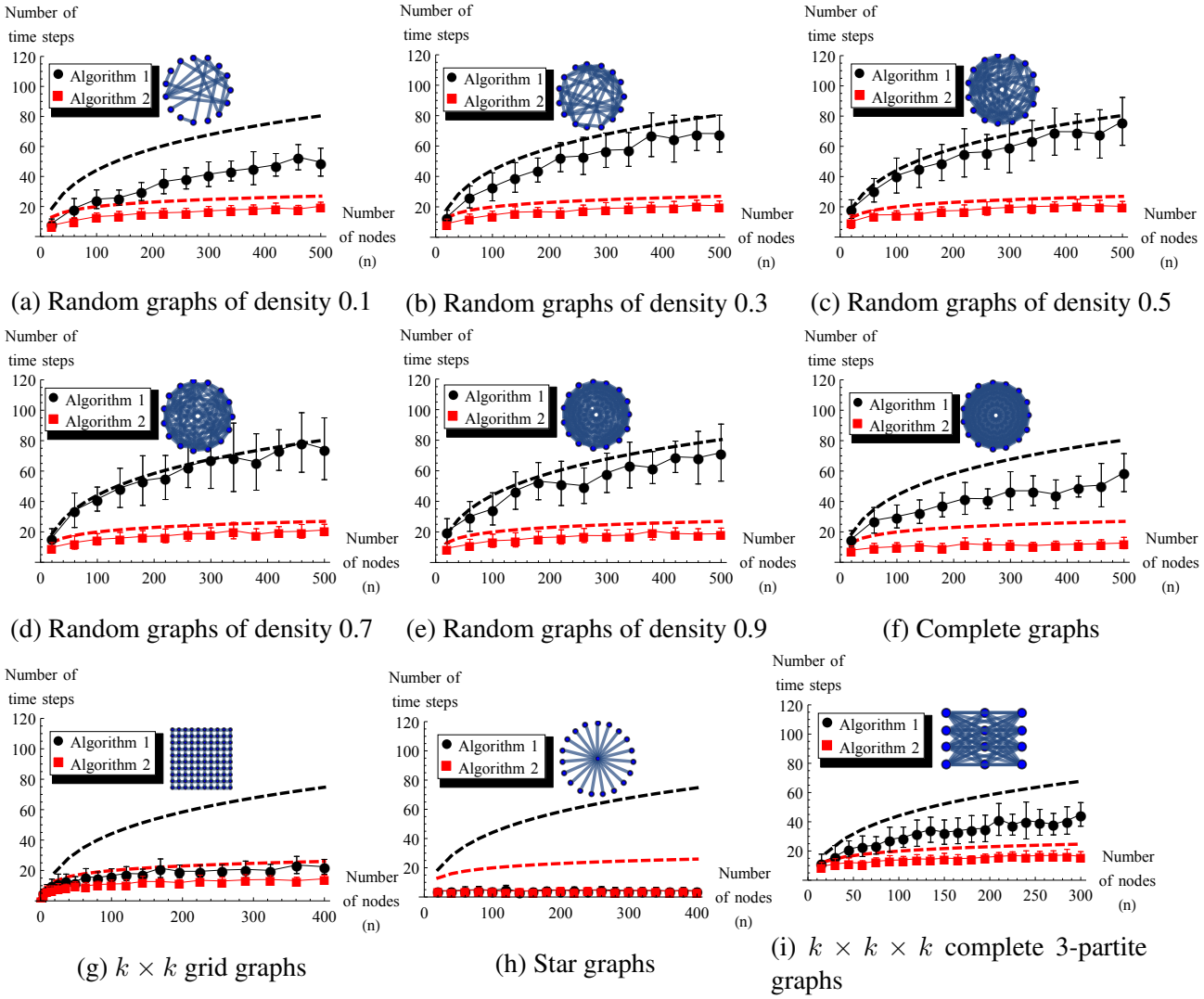
We then focus on the three versions of distributed greedy colouring algorithms we have proposed in Chapter 5 and Chapter 6 (Tables 5.2, 5.3 and 6.2). Again, we show experimentally the practical time and message complexity of these three algorithms on some standard families of graphs. Moreover, we also investigate the number of colours used on some standard graph colouring benchmarks.

7.1 Comparing different MIS selection algorithms

In this section, we compare the practical performance of the two different MIS selection algorithms summarised in Table 7.1.

Table 7.1: Comparison of the algorithms for distributed MIS selection

Algorithm	Time steps	Messages per node	Reference
Algorithm 1	$O(\log^2 n)$	Unknown	The algorithm described in Table 4.1
Algorithm 2	$O(\log n)$	$O(1)$	The algorithm[1] described in Table 6.1



(a) Random graphs of density 0.1 (b) Random graphs of density 0.3 (c) Random graphs of density 0.5

(d) Random graphs of density 0.7 (e) Random graphs of density 0.9 (f) Complete graphs

(g) $k \times k$ grid graphs (h) Star graphs (i) $k \times k \times k$ complete 3-partite graphs

Figure 7.1: Time complexity of the two MIS selection algorithms

7.1.1 Performance on running time

We first conducted experiments on random n-vertex graphs where each possible edge of the graph is present with probability den (referred to as the *density*). The value of n was varied from 20 to 500 (in steps of 40). The value of the density was set at 6 different values: 0.1, 0.3, 0.5, 0.7, 0.9, 1 (Note that $den = 1$ indicates the complete graph on n vertices). The running time was measured by the

[1]We set the local initial probability value $p = 1$ and the change factor for the probability value $f = 2$.

number of time steps (where two consecutive message exchanges in the algorithms represent one time step). For each combination of n and den, the computation of each algorithm was repeated 30 times. The mean value and the standard deviation of the running time of each run of the computation is shown in Figures 7.1a,7.1b,7.1c,7.1d,7.1e, and 7.1f respectively.

Similar experiments were also conducted for grid graphs, star graphs and complete 3-partite graphs. The value of k for $k \times k$ grid graphs was varied from 1 to 20 (in steps of 1). The value of n for an n-vertex star graph was varied from 10 to 400 (in steps of 20). The value of k for a $k \times k \times k$ complete 3-partite graphs was varied from 5 to 100 (in steps of 5). For each case, the computation of each algorithm was repeated 30 times and the running time was still measured by the number of time steps. The mean value and the standard deviation of the running time of each run of the computation is shown in Figures 7.1g, 7.1h and 7.1i, respectively.

To estimate the hidden constants in the upper bounds of $O(\log^2 n)$ for Algorithm 1 and $O(\log n)$ for Algorithm 2, the performance of both algorithms is compared with the function $\log_2^2 n$ shown as a black dashed line, and the function $3\log_2 n$ shown as a red dashed line, in Figure 7.1.

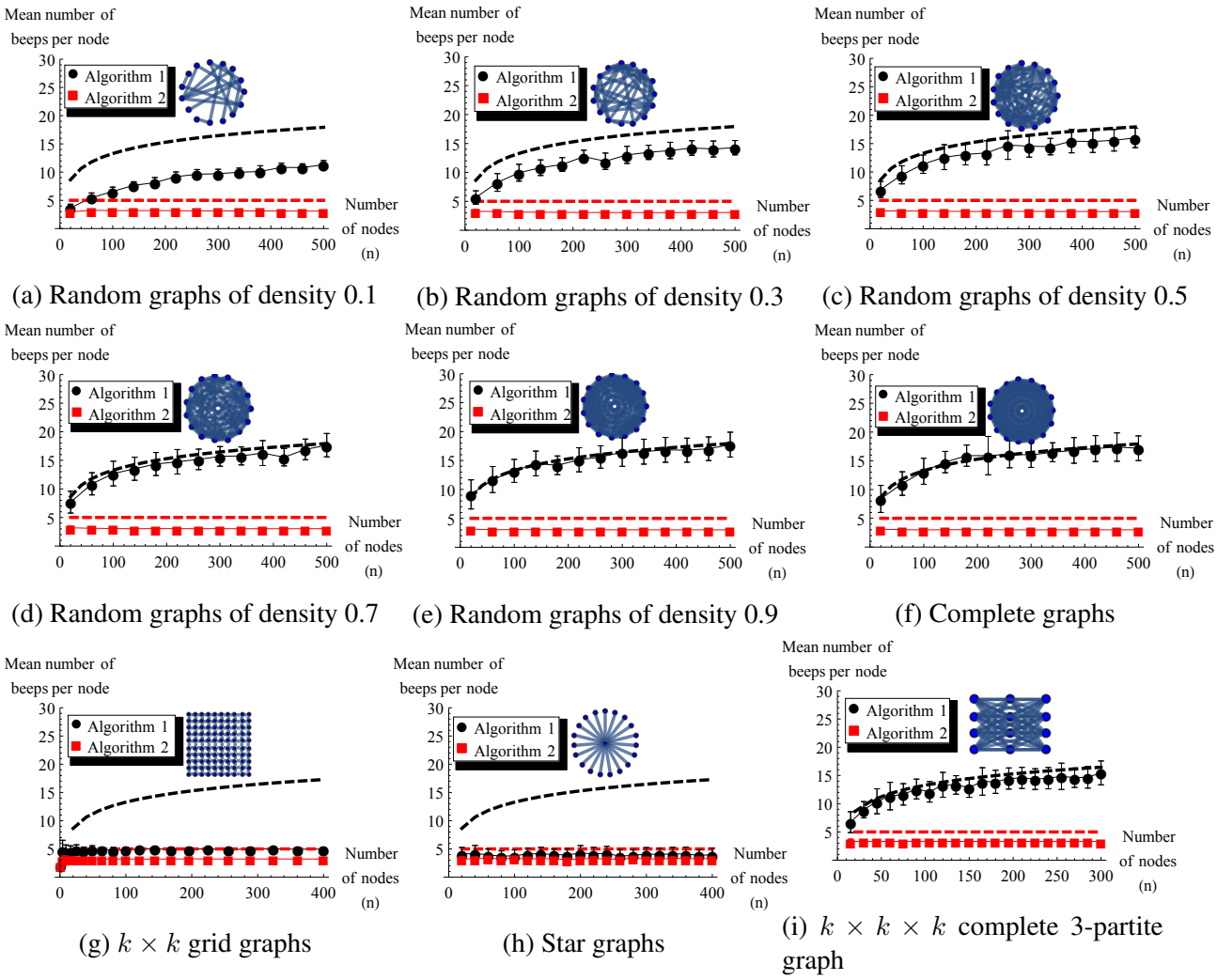
These results suggest that the mean value of the actual number of time steps needed to complete Algorithm 1 and Algorithm 2 is bounded tightly by the dashed lines, and even better performance of running time for both algorithms is achieved for grid graphs, star graphs, and complete 3-partite graphs.

7.1.2 Performance on messages sent by each node

We refer to the times when each vertex broadcasts a message to its neighbours as *beeps*. During the running time experiments described above, the mean number of beeps at each vertex was also recorded. The experimental results are shown in Figure 7.2. The results confirm Theorem 6.1.5 that the expected number of messages sent by each processor running Algorithm 2 is $O(1)$ for all types of graphs considered. More specifically, we found that the mean number of beeps sent by each node in Algorithm 2 is less than 4, regardless of the size of the graph. However, our random graph experiments indicated that the mean number of beeps sent by each node in Algorithm 1 increased with the size of the network. We compare the results against the function $2\log_2 n$ shown as a black dashed line, and the function 5 shown as a red dashed line, in Figure 7.2.

7.1.3 Performance on MaxIS selection (NP-hard) benchmarks

We also investigated the size of MIS chosen by both Algorithm 1 and Algorithm 2 on some standard maximum clique benchmark instances at `http://iridia.ulb.ac.be/~fmascia/maximum_`



(a) Random graphs of density 0.1 (b) Random graphs of density 0.3 (c) Random graphs of density 0.5

(d) Random graphs of density 0.7 (e) Random graphs of density 0.9 (f) Complete graphs

(g) $k \times k$ grid graphs (h) Star graphs (i) $k \times k \times k$ complete 3-partite graph

Figure 7.2: Message complexity of the two MIS selection algorithms

`clique/#detC125.9` and some standard maximum independent set (MaxIS) benchmark instances at `http://www.nlsde.buaa.edu.cn/~kexu/benchmarks/graph-benchmarks.htm`.

The maximum clique benchmark instances with file name .clq are selected from the Second DIMACS Implementation Challenge (1992-1993). Since a maximum independent set of a graph G is equal to the maximum clique of the complementary graph of G, we tested the MIS selection algorithms on the complementary graphs of the .clq benchmark instances. (The files for two types were found to be unreadable and had to be omitted). However, for those MaxIS benchmarks with file name .mis, we directly tested our algorithms on a selected list of them (Every first two instances in each type are selected). Thus, all the benchmark instances are divided into two classes: the class with file name .clq and the other class with file name .mis.

All these benchmark instances were tested using a simulation of Algorithms 1 and 2 (see Ta-

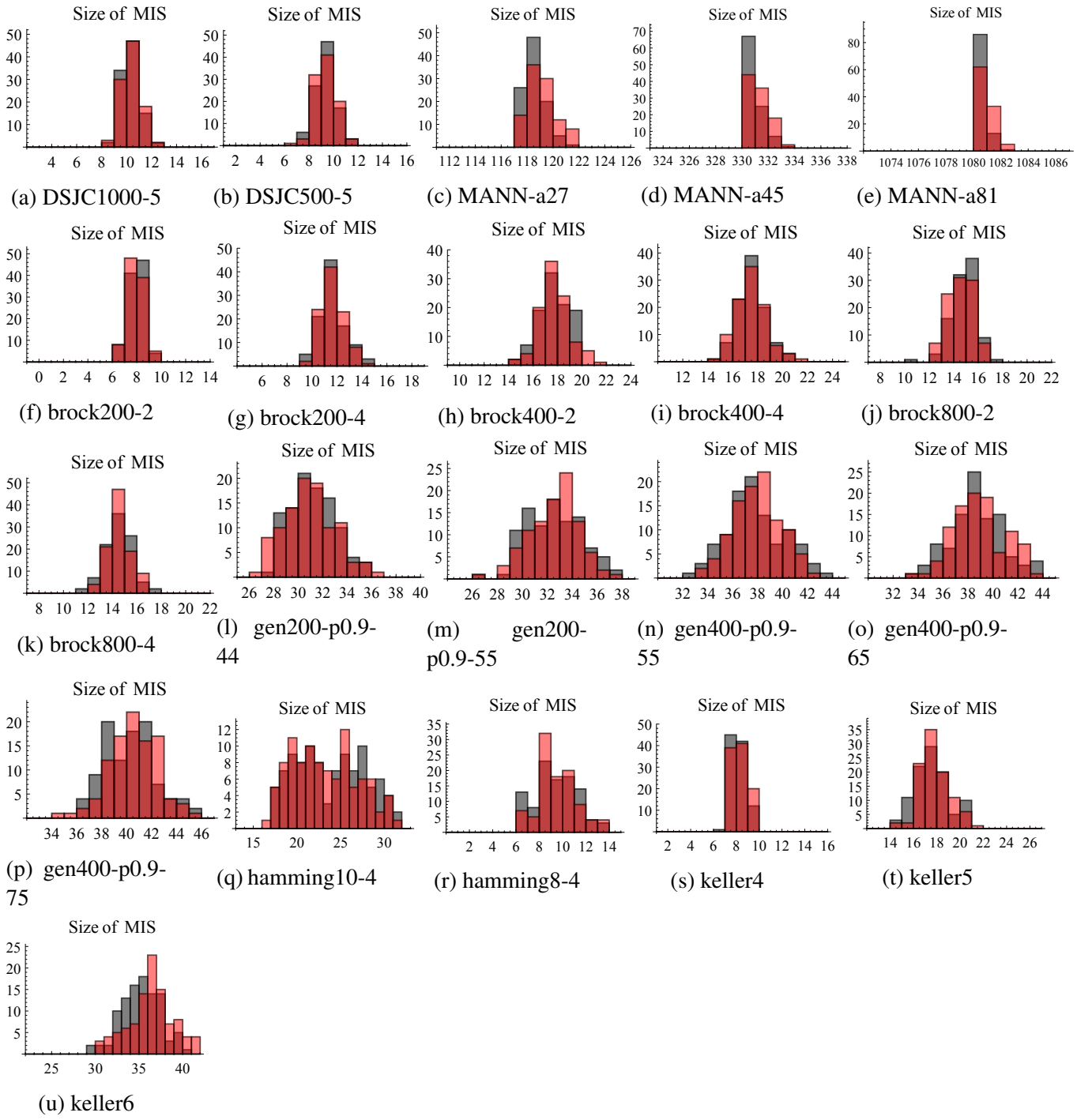


Figure 7.3: Size of MIS selected by the two MIS selection algorithms on .clq benmark instances over 100 runs

ble A.1 and Table A.2 for their source code), written using the *Mathematica* programming language, and running on a DELL desktop with an Intel(R) Core(TM) i5-2400 CPU running at 3.10GHz with 4GB of RAM. For each benchmark instance, the computations of Algorithm 1 and Algorithm 2 were

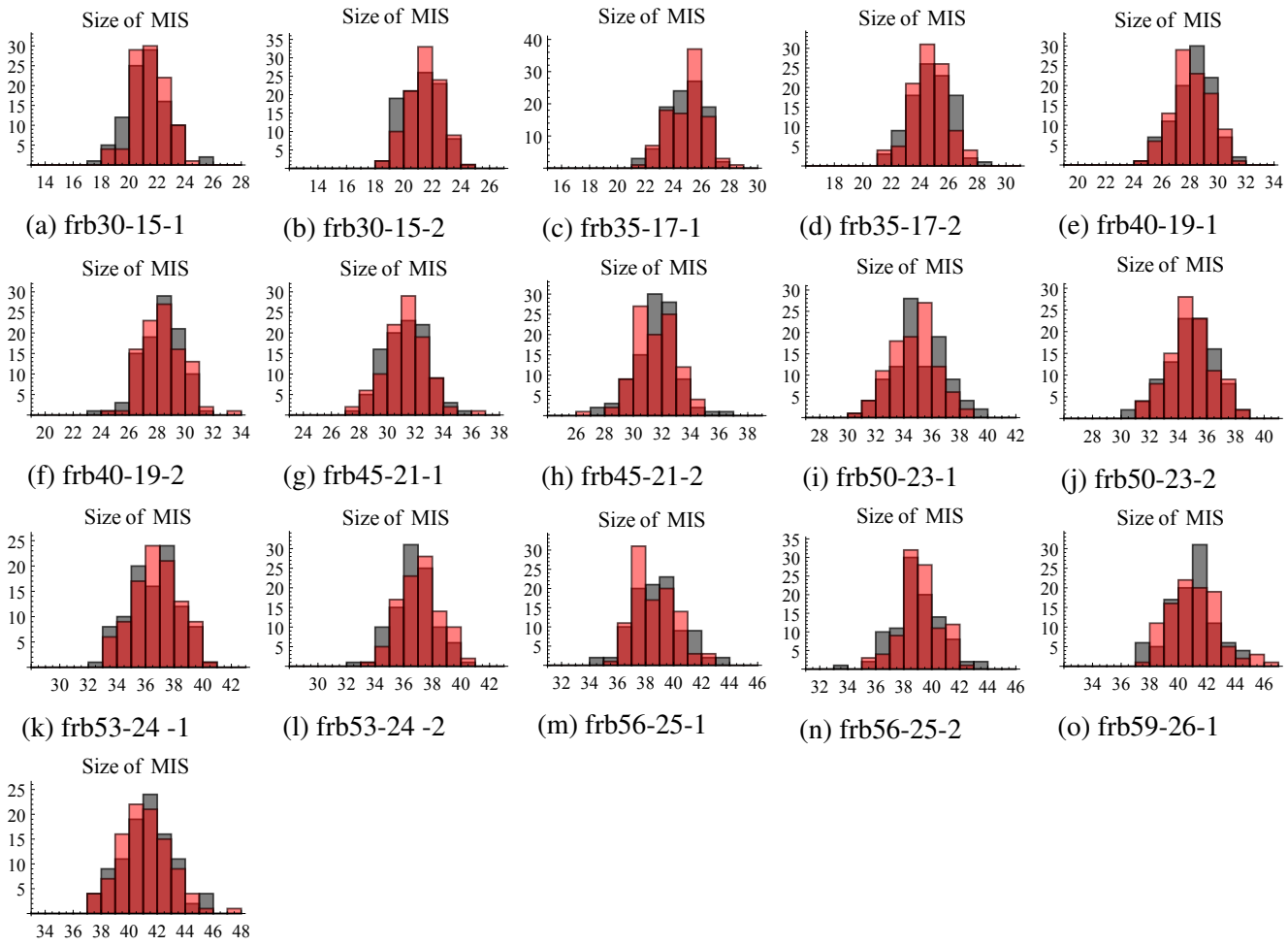


Figure 7.4: Size of MIS selected by the two MIS selection algorithms on .mis benchmark instances over 100 runs

each repeated 100 times and the size of MIS chosen, μ, in the resulting MIS was recorded for each run. The size of MIS chosen in each run is shown in the overlapped histograms in Figure 7.3 and in Figure 7.4 respectively for .clq instances and .mis instances (Colours key: Algorithm 1 in gray, Algorithm 2 in pale brown, overlap in dark brown). The results show that the distributions of the size of MIS chosen by the two algorithms are very similar and there is much overlapping.

A detailed summary of the results is shown in Table 7.2. We denote by $|V|$ and $|E|$ the number of nodes and the number of edges of the graph tested for MIS selection, and the maximum degree of the graph is denoted by Δ. The size of the maximum independent set is denoted by ω' for each instance. The minimum value $Min(\mu)$ and the maximum value $Max(\mu)$ for the MIS chosen on each instance over the 100 runs were recorded. We also compared Algorithm 1 and Algorithm 2 on the mean value of μ and its standard deviation over the 100 runs. Finally, we recorded the total

Table 7.2: Benchmark instances test on the two MIS selection algorithms

Benchmark instances of maximum independent set selection										
Type	Name	$\|V\|$	$\|E\|$	ω'	Δ	Algorithm	$Min(\mu)$	$Mean(\mu) \pm SD$	$Max(\mu)$	Time (seconds)
.clq	DSJC1000-5	1000	249674	15	552	1	8	9.81±0.787465	12	47.892307
						2	8	**9.86±0.816744**	12	**14.773295**
	DSJC500-5	500	62126	13	279	1	7	8.84±0.884433	11	13.556487
						2	6	**8.85±0.914253**	11	**5.116833**
	MANN-a27	378	702	126	13	1	117	118.07±0.86754	121	2.059213
						2	117	**118.64±1.11482**	121	**1.950012**
	MANN-a45	1000	249674	345	22	1	330	330.42±0.669388	333	6.208840
						2	330	**330.78±0.811284**	333	**5.803237**
	MANN-a81	3321	6480	≥ 1100	40	1	1080	1080.15±0.385992	1082	23.244149
						2	1080	**1080.43±0.590412**	1082	**21.387737**
	brock200-2	200	10024	12	121	1	6	**9.75±0.845368**	9	3.322821
						2	6	7.47±0.702880	9	**1.513210**
	brock200-4	200	6811	17	87	1	9	11.13±1.097790	14	2.839218
						2	9	**11.14±0.974628**	14	**1.435209**
	brock400-2	400	20014	29	125	1	14	17.20±1.263310	19	7.363247
						2	14	**17.25±1.290020**	21	**3.354021**
	brock400-4	400	20035	33	124	1	14	**17.03±1.167360**	20	7.144846
						2	14	17.00±1.271280	21	**3.338421**
	brock800-2	800	111434	24	327	1	10	**14.33±1.082970**	17	25.178561
						2	12	14.05±1.057680	16	**9.141659**
	brock800-4	800	111957	26	318	1	11	14.00±1.154700	17	24.897760
						2	12	**14.08±0.960640**	16	**8.938857**
	gen200-p0.9-44	200	1990	44	34	1	27	**30.68±1.852540**	35	1.668312
						2	26	30.43±2.109410	36	**1.138807**
	gen200-p0.9-55	200	1990	55	35	1	26	32.05±2.221910	37	1.748441
						2	26	**32.14±2.103480**	39	**1.232867**
	gen400-p0.9-55	400	7980	55	65	1	32	37.32±2.321960	43	4.929632
						2	33	**37.47±1.914620**	42	**2.683217**
	gen400-p0.9-65	400	7980	65	66	1	33	38.21±2.133400	43	4.976432
						2	33	**38.39±2.064100**	43	**2.636417**
	gen400-p0.9-75	400	7980	75	64	1	36	39.77±2.083390	45	5.007632
						2	34	**40.07±1.975890**	45	**2.714417**
	hamming10-4	1024	89600	40	175	1	17	**23.38±3.938360**	31	25.443763
						2	16	22.74±3.754110	31	**10.062064**
	hamming8-4	256	11776	16	92	1	6	8.92±1.823970	13	3.993626
						2	6	**9.10±1.795050**	**16**	**1.950012**
	keller4	171	5100	11	68	1	6	7.65±0.701729	9	2.028013
						2	7	**7.81±0.747994**	9	**1.107607**
	keller5	776	74710	27	215	1	14	17.07±1.499190	20	18.423718
						2	14	**17.31±1.323410**	21	**7.503648**
	keller6	3361	1026582	≥ 59	670	1	29	34.74±2.263550	40	207.434530
						2	30	**35.85±2.583460**	41	**62.416000**
.mis	frb30-15-1	450	17827	30	122	1	17	20.76±1.477920	25	7.394447
						2	18	**20.96±1.238440**	24	**3.619223**
	frb30-15-2	450	17874	30	116	1	18	20.77±1.324630	24	7.503648
						2	18	**20.98±1.222520**	24	**3.603623**
	frb35-17-1	595	27856	35	132	1	21	24.31±1.353600	27	11.232072
						2	21	**24.47±1.336780**	28	**5.038832**
	frb35-17-2	595	27847	35	134	1	21	**24.24±1.436330**	28	10.904470
						2	21	24.13±1.323100	27	**4.992032**
	frb40-19-1	760	41314	40	178	1	24	**27.75±1.438120**	31	15.350498
						2	24	27.62±1.419770	31	**6.708043**
	frb40-19-2	760	41263	40	171	1	23	27.77±1.462360	31	15.178897
						2	24	**27.91±1.498110**	33	**6.910844**
	frb45-21-1	945	59186	45	188	1	27	**30.82±1.565930**	35	20.794933
						2	27	30.80±1.563470	36	**9.016858**
	frb45-21-2	945	58624	45	191	1	27	**31.11±1.530140**	36	20.779333
						2	26	31.05±1.472820	34	**8.970057**
	frb50-23-1	1150	80072	50	208	1	30	**34.57±1.881620**	39	27.393776
						2	30	34.21±1.653250	38	**11.762475**
	frb50-23-2	1150	80851	50	227	1	30	34.36±1.726330	38	26.535770
						2	31	**34.37±1.586910**	38	**11.809276**
	frb53-24 -1	1272	94227	53	232	1	32	36.08±1.762000	40	31.605803
						2	33	**36.25±1.653740**	40	**13.416086**
	frb53-24-2	1272	94289	53	232	1	32	36.25±1.452100	40	31.683803
						2	33	**36.64±1.460040**	40	**13.026084**
	frb56-25-1	1272	94289	56	232	1	34	**38.32±1.802800**	43	36.161032
						2	35	38.13±1.535140	42	**14.851295**
	frb56-25-2	1400	109401	56	237	1	33	38.48±1.743560	43	36.176632
						2	35	**38.64±1.446140**	42	**15.085297**
	frb59-26-1	1534	126555	59	277	1	37	40.42±1.640280	44	41.293465
						2	37	**40.61±1.786000**	46	**16.645307**
	frb59-26-2	1534	126163	59	265	1	37	**40.76±1.864720**	45	42.198270
						2	37	40.64±1.850580	47	**16.832508**

running time for the 100 runs.

The results in Table 7.2 show that neither Algorithm 1 nor Algorithm 2 performed well on maximizing the size of MIS chosen. To compare the effectiveness of the two algorithms, in Table 7.2 the larger of the two values for $Mean(\mu)$ are highlighted in bold, and the shorter of the two total running times are also highlighted in bold. The results show that Algorithm 2 performed slightly better than Algorithm 1 in terms of the size of MIS chosen (13 results in bold for Algorithm 1 and 24 results in bold for Algorithm 2). Moreover, both algorithms achieved these values with a very short running time. Overall Algorithm 2 did better than Algorithm 1 on the running time over 100 runs (all the 37 results are in bold for Algorithm 2).

7.2 Comparing different greedy colouring algorithms

In this section, we compare the practical performance of the three different greedy colouring algorithms summarised in Table 7.3.

Table 7.3: Comparison of the algorithms for distributed greedy colouring

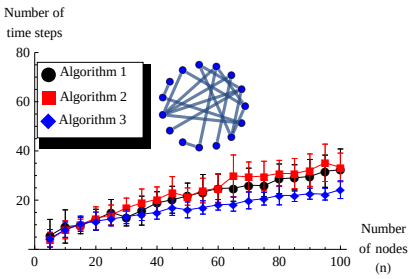
Algorithm	Time steps	Messages per node	Reference
Algorithm 1	$O(\Delta^2 \log^2 n)$	$O(\Delta \log n)$	The algorithm described in Table 5.2
Algorithm 2	$O(\Delta \log^2 n)$	$O(\Delta \log n)$	The algorithm described in Table 5.3
Algorithm 3	$O(\Delta + \log n)$	$O(1)$	The algorithm[2] described in Table 6.2

7.2.1 Performance on running time

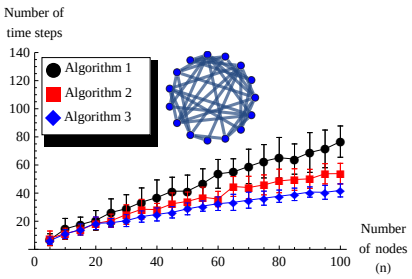
We again first conducted experiments on random n-vertex graphs where each possible edge of the graph is present with probability den (referred to as the *density*). The value of n was varied from 5 to 100 (in steps of 5). The value of the density was set at 6 different values: 0.1, 0.3, 0.5, 0.7, 0.9, 1 (with $den = 1$ indicating the complete graph on n vertices). The running time was measured by the number of time steps (where two consecutive message exchanges in the algorithms represent one time step). For each combination of n and den, the computation of each algorithm was repeated 30 times. The mean value and the standard deviation of the running time of each run of the computation is shown in Figures 7.1a,7.1b,7.1c,7.1d,7.1e, and 7.1f respectively.

Among the random graph experiments, the worst case occurs with complete graphs, as suggested by the proofs of Theorem 5.3.1, Theorem 5.3.3 and Theorem 6.2.1. The mean value of the actual number of time steps needed to complete Algorithm 1 (respectively Algorithm 2) at all vertices appears to be growing much more slowly than $\Delta^2 \log^2 n$ (respectively $\Delta \log^2 n$) across a wide range

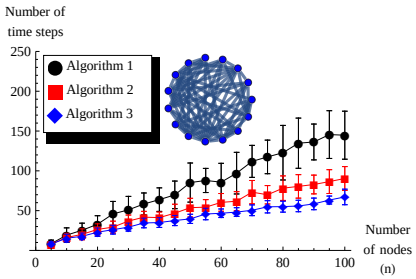
[2]We set the local initial probability value $p = 1$ and the change factor for the probability value $f = 2$.



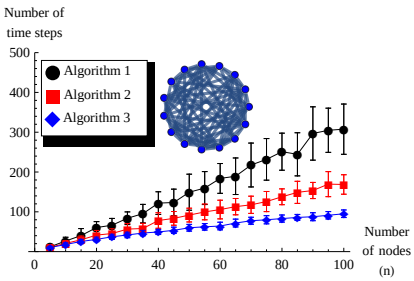
(a) Random graphs of density 0.1



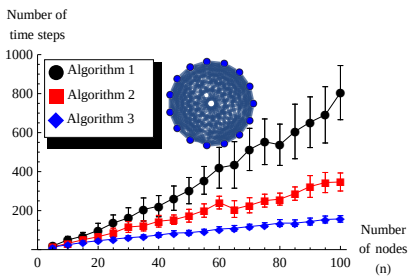
(b) Random graphs of density 0.3



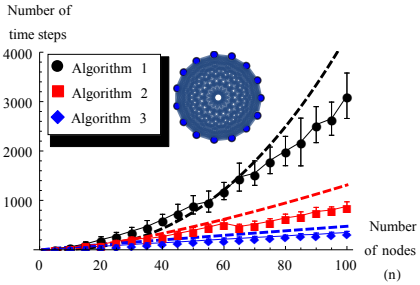
(c) Random graphs of density 0.5



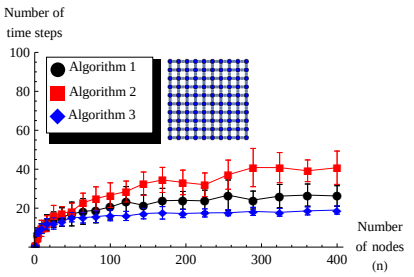
(d) Random graphs of density 0.7



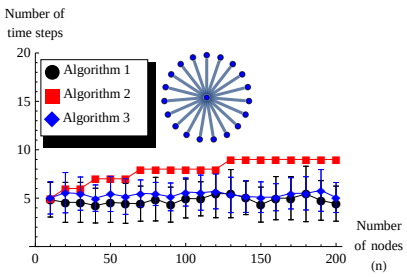
(e) Random graphs of density 0.9



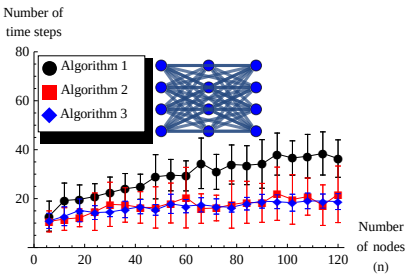
(f) Complete graphs



(g) $k \times k$ grid graphs



(h) Star graphs



(i) $k \times k \times k$ complete 3-partite graphs

Figure 7.5: Time complexity of the three greedy colouring algorithms

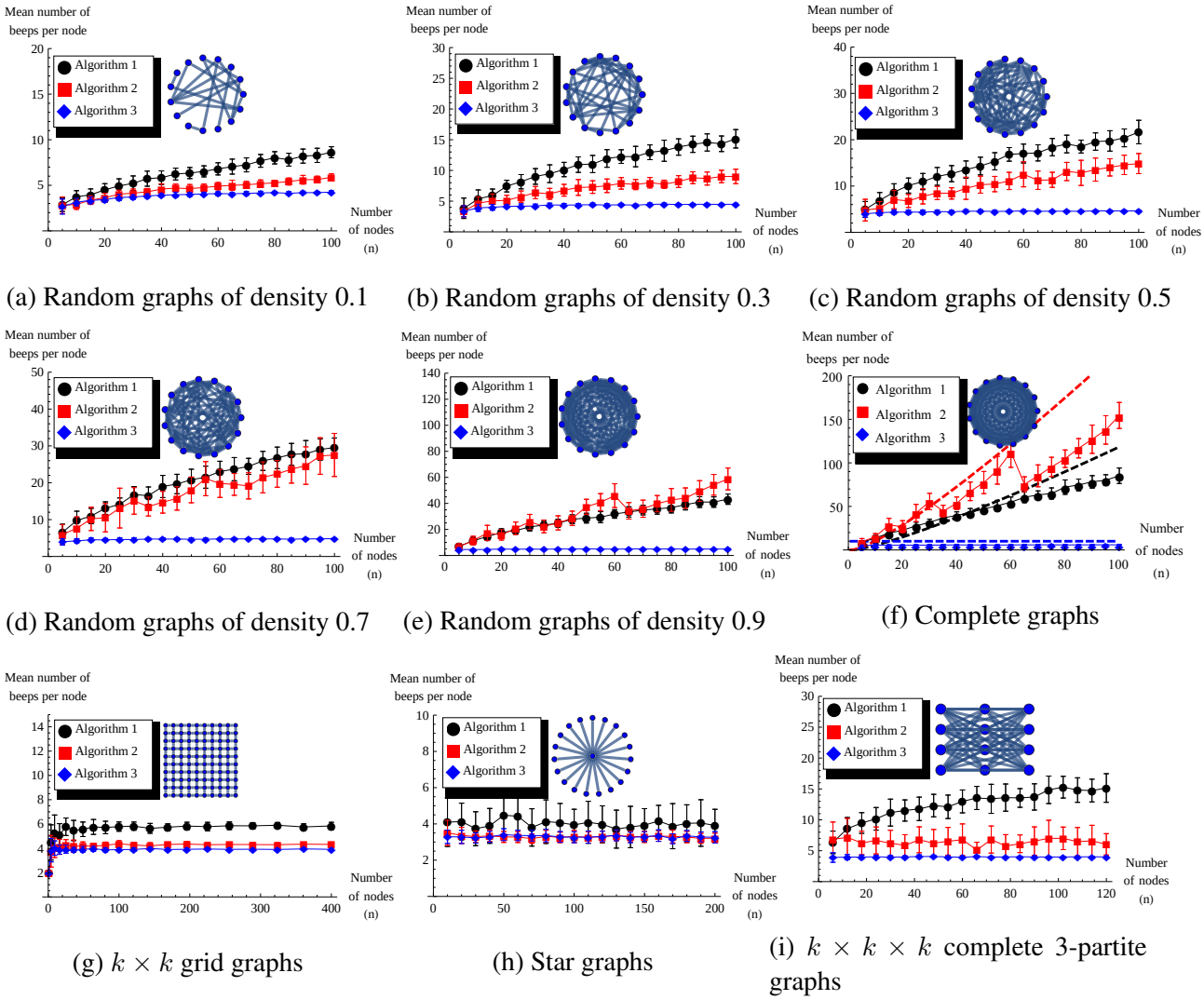
densities. The mean value of the actual number of time steps needed to complete Algorithm 3 appears to be growing much more slowly than the other two algorithms across a wide range densities. To estimate the hidden constants in the upper bounds of $O(\Delta^2 \log^2 n)$ for Algorithm 1, $O(\Delta \log^2 n)$ for Algorithm 2 and $O(\Delta + \log n)$ for Algorithm 3, the performance of the three algorithms is compared with the function $0.01(n-1)^2 \log_2^2 n$ shown as a black dashed line, the function $0.3(n-1)\log_2^2 n$ shown as a red dashed line, and the function $4.5((n-1)+\log_2 n)$ shown as a blue dashed line in Figure 7.5f.

Similar experiments were also conducted for grid graphs, star graphs and complete 3-partite graphs. The value of k for $k \times k$ grid graphs was varied from 1 to 20 (in steps of 1). The value of n for an n-vertex star graph was varied from 10 to 200 (in steps of 20). The value of k for a $k \times k \times k$ complete 3-partite graphs was varied from 2 to 40 (in steps of 2). For each case, the computation

of each algorithm was repeated 30 times and the running time was measured by the number of time steps. The mean value and the standard deviation of the running time of each run of the computation is shown in Figures 7.5g, 7.5h and 7.5i, respectively.

These results suggest that even better performance of running time for the three algorithms is achieved for grid graphs, star graphs, and complete 3-partite graphs, where the running time appears to grow no faster than $\log n$ for any of the algorithms.

7.2.2 Performance on messages sent by each node



(a) Random graphs of density 0.1

(b) Random graphs of density 0.3

(c) Random graphs of density 0.5

(d) Random graphs of density 0.7

(e) Random graphs of density 0.9

(f) Complete graphs

(g) $k \times k$ grid graphs

(h) Star graphs

(i) $k \times k \times k$ complete 3-partite graphs

Figure 7.6: Message complexity of the three greedy colouring algorithms

We now refer to the times when each vertex broadcasts a colour value to its neighbours as *beeps*. During the running time experiments described above, the mean number of beeps at each vertex was

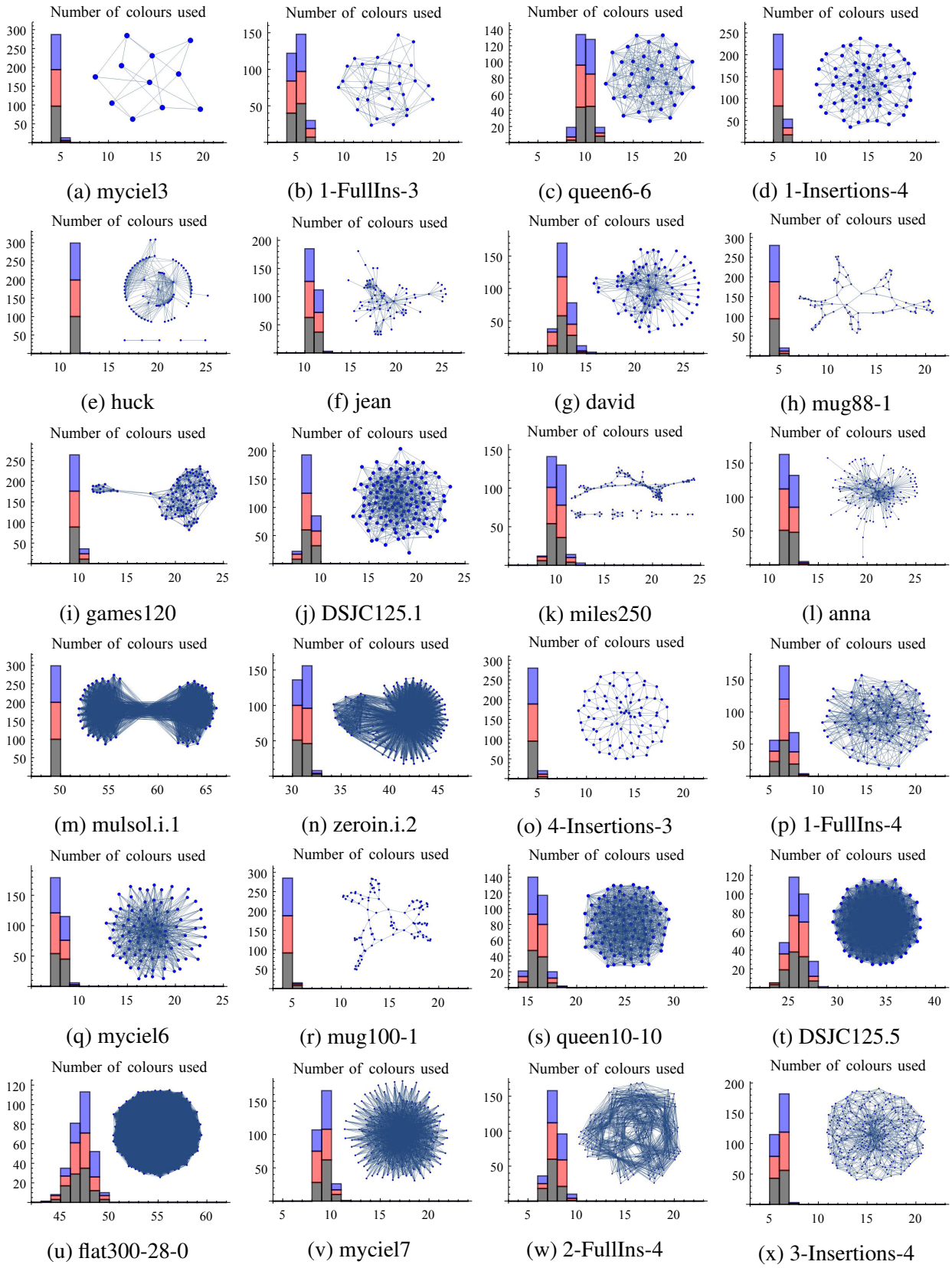
also recorded. The experimental results are shown in Figure 7.6. The results confirm Theorem 5.3.2, Theorem 5.3.4 and Theorem 6.2.5 that the expected number of messages sent by each processor running Algorithm 1 or Algorithm 2 (respectively Algorithm 3) is $O(\Delta \log n)$ (respectively $O(1)$) for all types of graphs considered. To estimate the hidden constants in the bound of $O(\Delta \log n)$ and $O(1)$, number of beeps is compared against the function $0.18(n-1)\log_2 n$ shown as a black dashed line, the function $0.35(n-1)\log_2 n$ shown as a red dashed line, and the function 10 shown as a blue dashed line in Figure 7.6f.

7.2.3 Performance on graph colouring (NP-hard) benchmarks

We also investigated the number of colours used by the three algorithms. Since they all construct a greedy colouring, the number of colours used is at most $\Delta + 1$. However, we expect that in many cases the number of colours will be much lower than this, and in some cases may even match the optimal number of colours, χ.

Algorithm 1, Algorithm 2 and Algorithm 3 were tested on some of the standard benchmark instances for graph colouring provided at `https://sites.google.com/site/graphcoloring/vertex-coloring`. These benchmark instances on this site are divided into four classes indicating their estimated hardness. All instances for which no polynomial time algorithm is known are classified as **NP**, and the letter after this identifier indicates how long it takes to solve the problem using state-of-the-art software. According to the documentation accompanying these benchmark instances, **NP**-s, **NP**-m, and **NP**-h indicate instances which can be solved in a few seconds (less than a minute), in a few minutes (less than an hour) and in a few hours (less than a day), respectively. Finally, **NP**-? means the instance is not solved or the time is not known. (If the chromatic number is given for instances of this final type, then it is known by construction.) All the standard benchmark instances with known chromatic numbers can be divided into three classes by their sizes: those with at most 300 vertices, those with more than 300 but fewer than 700 vertices, and those with 700 or more vertices. From each of these classes, we took one instance of each type of benchmark available at the site. (The files for three types containing r125.1, qg.order60, and wap05a were found to be unreadable and had to be omitted).

All these benchmark instances were tested using a simulation of Algorithms 1, 2 and 3 (see Table A.3 and Table A.4 for their source code), written using the *Mathematica* programming language, and running on a DELL desktop with an Intel(R) Core(TM) i5-2400 CPU running at 3.10GHz with 4GB of RAM. For each benchmark instance, the computations of Algorithm 1, Algorithm 2 and Algorithm 3 were each repeated 100 times and the number of distinct colours, γ, in the resulting colouring was recorded for each run. The number of colours used in each run is shown in the stacked histograms in Figures 7.7, 7.8, 7.9 (Colours key: Algorithm 1 in gray, Algorithm 2 in pale



(a) myciel3 (b) 1-FullIns-3 (c) queen6-6 (d) 1-Insertions-4

(e) huck (f) jean (g) david (h) mug88-1

(i) games120 (j) DSJC125.1 (k) miles250 (l) anna

(m) mulsol.i.1 (n) zeroin.i.2 (o) 4-Insertions-3 (p) 1-FullIns-4

(q) myciel6 (r) mug100-1 (s) queen10-10 (t) DSJC125.5

(u) flat300-28-0 (v) myciel7 (w) 2-FullIns-4 (x) 3-Insertions-4

Figure 7.7: Number of colours used by the three greedy colouring algorithms on smaller benchmark instances ($n \leq 300$) over 100 runs

Table 7.4: Smaller benchmark instances ($n \leq 300$) test on the three greedy colouring algorithms

Benchmark instances of graph colouring										
Hardness	Names (.col)	$\|V\|$	$\|E\|$	χ	$\Delta+1$	Algorithms	$Min(\gamma)$	$Mean(\gamma) \pm SD$	$Max(\gamma)$	Time (seconds)
NP-s	myciel3	11	20	4	6	1	**4**	**4.03±0.171447**	5	0.249602
						2	**4**	**4.03±0.171447**	5	**0.171601**
						3	**4**	4.07±0.256432	5	0.312002
	1-FullIns-3	30	100	4	12	1	**4**	**4.67±0.603943**	6	1.216808
						2	**4**	4.68±0.679869	6	**0.951606**
						3	**4**	4.73±0.649086	6	1.388409
	queen6-6	36	290	7	20	1	8	9.58±0.684312	11	3.915625
						2	8	**9.44±0.640707**	11	**2.652017**
						3	8	9.45±0.796140	11	3.525623
	1-Insertions-4	67	232	5	23	1	**5**	5.17±0.377525	6	**4.960832**
						2	**5**	**5.16±0.368453**	6	5.319634
						3	**5**	5.20±0.402015	6	5.148033
	huck	74	301	11	54	1	**11**	**11.00±0.000000**	**11**	14.757695
						2	**11**	11.01±0.100000	12	13.462886
						3	**11**	**11.00±0.000000**	**11**	**11.918476**
	jean	80	254	10	37	1	**10**	**10.37±0.485237**	11	13.228885
						2	**10**	10.37±0.505625	12	12.573681
						3	**10**	10.44±0.537860	12	**11.684475**
	david	87	406	11	83	1	**11**	12.20±0.666667	14	18.142916
						2	**11**	**12.00±0.681650**	14	16.754507
						3	**11**	12.50±0.797724	15	**15.787301**
	mug88-1	88	146	4	5	1	**4**	**4.06±0.238683**	5	9.188459
						2	**4**	**4.06±0.238683**	5	9.937264
						3	**4**	4.08±0.272660	5	**8.424054**
	games120	120	638	9	14	1	**9**	**9.11±0.314466**	10	35.303026
						2	**9**	9.13±0.337998	10	30.014592
						3	**9**	9.12±0.326599	10	**25.490563**
	DSJC125.1	125	736	5	24	1	7	8.24±0.588097	9	26.005367
						2	7	**8.17±0.569512**	9	24.757359
						3	7	8.22±0.523778	9	**22.323743**
	miles250	128	387	8	17	1	**8**	**9.38±0.663325**	11	34.881824
						2	**8**	9.49±0.688946	11	30.357795
						3	**8**	9.68±0.708961	12	**26.020967**
	anna	138	493	11	72	1	**11**	11.50±0.522233	13	40.388659
						2	**11**	**11.41±0.533617**	13	41.277865
						3	**11**	11.51±0.541136	13	**31.403001**
	mulsol.i.1	197	3925	49	122	1	**49**	**49.00±0.000000**	**49**	568.374043
						2	**49**	**49.00±0.000000**	**49**	219.977010
						3	**49**	49.01±0.100000	50	**162.958645**
	zeroin.i.2	211	3541	30	141	1	**30**	30.52±0.559220	32	295.746696
						2	**30**	**30.52±0.521846**	32	163.676249
						3	**30**	30.68±0.548276	32	**121.072376**
NP-m	4-Insertions-3	79	156	4	14	1	**4**	**4.05±0.219043**	5	6.411641
						2	**4**	4.06±0.238683	5	6.879644
						3	**4**	4.09±0.287623	5	**5.912438**
	1-FullIns-4	93	593	5	33	1	**5**	**6.00±0.710669**	8	10.998071
						2	**5**	6.05±0.625631	8	10.701669
						3	**5**	6.15±0.701729	8	**10.218065**
	myciel6	95	755	7	48	1	**7**	7.47±0.521362	9	12.682881
						2	**7**	**7.35±0.519810**	9	**12.355279**
						3	**7**	7.45±0.557320	9	12.573681
	mug100-1	100	166	4	5	1	**4**	4.08±0.272660	5	11.871676
						2	**4**	4.04±0.196946	5	13.119684
						3	**4**	**4.03±0.171447**	5	**10.467667**
NP-h	queen10-10	100	2940	11	36	1	14	15.47±0.758188	18	38.953450
						2	14	**15.46±0.716614**	17	**26.254968**
						3	14	15.49±0.784895	18	27.253375
	DSJC125.5	125	3891	17	76	1	23	**25.22±0.938298**	27	128.919226
						2	23	25.26±0.871780	27	73.804073
						3	24	25.53±0.936952	28	**65.551620**
	flat300-28-0	300	21695	28	163	1	43	**46.42±1.147500**	49	1661.644652
						2	44	46.58±1.102610	49	816.259632
						3	44	46.93±0.997522	49	**583.365740**
NP-?	myciel7	191	2360	8	96	1	**8**	8.82±0.592546	10	57.938771
						2	**8**	**8.60±0.619547**	10	56.706364
						3	**8**	8.79±0.640312	11	**48.984314**
	2-FullIns-4	212	1621	6	56	1	**6**	**7.05±0.657129**	9	66.316025
						2	**6**	7.37±0.661419	9	65.645221
						3	**6**	7.38±0.762505	9	**52.104334**
	3-Insertions-4	281	1046	5	57	1	**5**	**5.58±0.516006**	7	96.798620
						2	**5**	5.65±0.500000	7	126.766413
						3	**5**	5.65±0.500000	7	**78.125301**

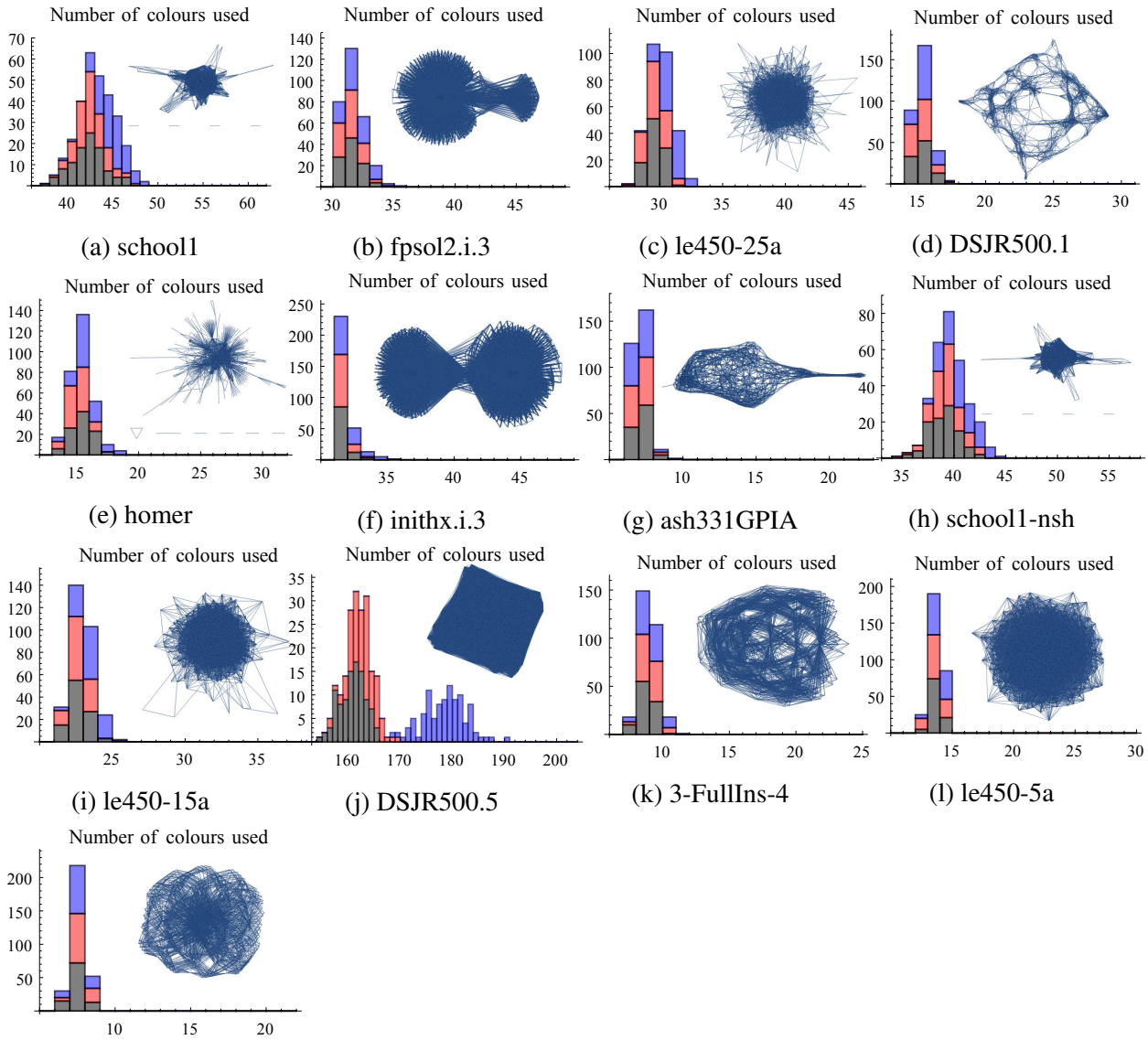
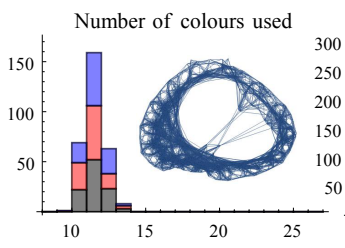


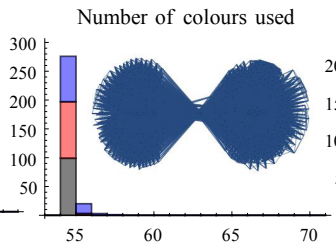
Figure 7.8: Number of colours used by the three greedy colouring algorithms on medium-sized benchmark instances ($300 < n < 700$) over 100 runs

brown and Algorithm 3 in light blue). The results show that the distributions of the number of colours used by the three algorithms are very similar and there is much overlapping. A summary of the results is shown in Tables 7.4, 7.5, 7.6. To our surprise, for 26 of the 42 benchmark instances the minimum value of γ found during the 100 runs was equal to the chromatic number χ for all the three algorithms. These values are shown in bold in Tables 7.4, 7.5, 7.6. We also compared Algorithm 1, Algorithm 2 and Algorithm 3 on the mean value of γ and its standard deviation over the 100 runs. Finally, we recorded the total running time for the 100 runs.

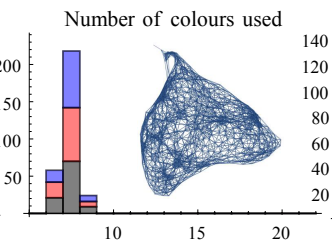
To compare the effectiveness of the three algorithms, in Tables 7.4, 7.4, 7.4 the smallest of the



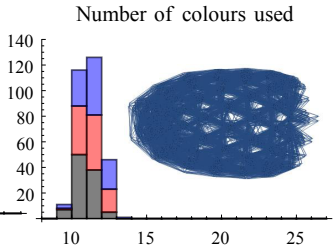
(a) will199GPIA



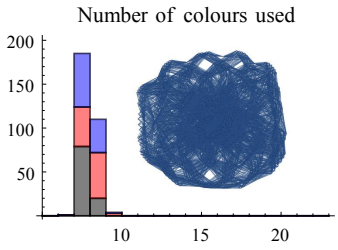
(b) inithx.i.1



(c) ash608GPIA



(d) 5-FullIns-4



(e) 3-Insertions-5

Figure 7.9: Number of colours used by the three greedy colouring algorithms on larger benchmark instances ($n \geq 700$) over 100 runs

three values for $Mean(\gamma)$ are highlighted in red, and the shortest of the three total running times are highlighted in blue. The results show that Algorithm 1 and Algorithm 2 outperformed Algorithm 3 and performed very similarly in terms of the numbers of colours used (25 red cells for Algorithm 1, 18 red cells for Algorithm 2 and 3 red cells for Algorithm 3). Moreover, all the three algorithms achieved these values with a very short running time. Overall Algorithm 3 did better than Algorithm 1 and Algorithm 2 on the running time over 100 runs (1 blue cells for Algorithm 1, 5 blue cells for Algorithm 2 and 36 blue cells for Algorithm 3). Even for the benchmark instances which were described as being in the hardest class, all the algorithms performed very well on both minimising the number of colours used and the running time.

7.3 Summary

In this chapter, we have systematically examined the practical performance of the distributed MIS selection algorithms and greedy colouring algorithms we have theoretically investigated in previous chapters. The MIS selection algorithms were not only tested on some standard families of graphs but also tested on some standard maximum clique and maximum independent set benchmarks to show their performance on maximizing the size of MIS chosen. Similarly, the greedy algorithms were not only tested on some standard families of graphs but also tested on some standard graph colouring benchmarks to show their performance on minimizing the number of colours used.

Though the MIS selection algorithms did not perform very well on maximizing the size of MIS,

Table 7.5: Medium-sized benchmark instances ($300 < n < 700$) test on the three greedy colouring algorithms

Benchmark instances of graph colouring										
Hardness	Names (.col)	$\|V\|$	$\|E\|$	χ	$\Delta+1$	Algorithms	$Min(\gamma)$	$Mean(\gamma) \pm SD$	$Max(\gamma)$	Time (seconds)
NP-s	school1	385	19095	14	283	1	37	**41.75±1.914190**	46	2104.328689
						2	38	42.05±1.641480	47	1234.825916
						3	39	44.32±1.581970	48	**878.878434**
	fpsol2.i.3	425	8688	30	347	1	**30**	31.02±0.816249	33	1589.899792
						2	**30**	**30.96±0.851855**	34	937.737611
						3	**30**	31.41±1.073980	35	**605.985884**
	le450-25a	450	8260	25	129	1	27	**29.11±0.737111**	31	1283.623028
						2	27	29.13±0.860526	31	1017.204520
						3	28	30.33±0.817177	32	**789.755063**
	DSJR500.1	500	3555	12	26	1	14	14.84±0.720830	17	1101.694662
						2	14	**14.73±0.679498**	17	938.002813
						3	14	15.02±0.619221	17	**602.881465**
	homer	561	1628	13	100	1	**13**	14.91±0.922174	17	914.165860
						2	**13**	**14.54±0.757721**	16	1024.614568
						3	**13**	15.24±1.045630	18	**613.021530**
	inithx.i.3	621	13969	31	543	1	**31**	**31.18±0.457927**	33	3877.654457
						2	**31**	31.21±0.573752	35	2459.496166
						3	**31**	31.57±0.843933	34	**1421.356311**
	ash331GPIA	662	4181	4	24	1	6	6.72±0.604361	9	719.024209
						2	6	6.58±0.553775	8	880.220042
						3	6	**6.57±0.555141**	8	**524.069759**
NP-m	school1-nsh	352	14612	14	233	1	35	**38.51±1.431920**	42	1433.976792
						2	34	38.77±1.455430	42	920.000297
						3	37	40.07±1.590720	44	**676.045934**
	le450-15a	450	8168	15	100	1	21	**22.18±0.716050**	24	872.279592
						2	21	22.19±0.691872	25	757.275654
						3	21	22.89±0.802710	25	**602.163860**
NP-h	DSJR500.5	500	58862	122	389	1	154	**160.31±2.481020**	165	56648.331128
						2	156	162.28±2.412430	169	11788.449566
						3	168	178.06±4.059710	190	**6750.490872**
NP-?	3-FullIns-4	405	3524	7	85	1	**7**	**8.26±0.645419**	10	266.496508
						2	**7**	8.51±0.658971	10	311.986400
						3	**7**	8.58±0.793662	11	**208.479736**
	le450-5a	450	5714	5	43	1	12	13.16±0.486588	14	564.442818
						2	12	**13.10±0.627646**	14	539.857061
						3	12	13.34±0.572431	14	**393.481322**
	2-Insertions-5	597	3936	6	150	1	**6**	**6.98±0.531436**	8	558.623981
						2	**6**	7.16±0.486588	8	717.885402
						3	**6**	7.08±0.525703	8	**408.582219**

Table 7.6: Larger benchmark instances ($n \geq 700$) test on the three greedy colouring algorithms

Benchmark instances of graph colouring										
Hardness	Names (.col)	$\|V\|$	$\|E\|$	χ	$\Delta+1$	Algorithms	$Min(\gamma)$	$Mean(\gamma) \pm SD$	$Max(\gamma)$	Time (seconds)
NP-s	will199GPIA	701	6772	7	39	1	10	11.07±0.755518	13	1202.892511
						2	9	**10.92±0.761179**	13	1332.310940
						3	10	11.09±0.726066	13	**805.323962**
	inithx.i.1	864	18707	54	503	1	**54**	**54.01±0.100000**	55	8690.176106
						2	**54**	54.02±0.140705	55	3773.632990
						3	**54**	54.26±0.561743	57	**2141.144925**
NP-m	ash608GPIA	1216	7844	4	21	1	6	6.88±0.537108	8	2658.647043
						2	6	**6.86±0.512865**	8	3606.758720
						3	6	6.92±0.485757	8	**1865.678359**
NP-?	5-FullIns-4	1085	11395	9	161	1	**9**	**10.41±0.697687**	12	2621.440804
						2	**9**	10.78±0.746439	12	3336.970591
						3	**9**	10.91±0.817671	13	**1828.830923**
	3-Insertions-5	1406	9695	6	282	1	**6**	**7.19±0.419114**	8	3528.212217
						2	7	7.58±0.553775	9	4807.358016
						3	7	7.40±0.512471	9	**2426.221153**

the number of colours used by the distributed greedy colouring algorithms turned out to be optimal or near-optimal for many standard graph colouring benchmarks. It is worth noting that our distributed MIS selection and greedy colouring algorithms are not designed to find a largest MIS or minimize the number of colours used (and hence offer no proof or guarantee), however, the experimental

results from the benchmark instance tests show that the distributed greedy colouring algorithms provide effective heuristic approaches to computing a colouring with a small number of colours.

CHAPTER 8

Conclusions and open problems

8.1 Conclusions

In this thesis, we have proposed and analysed some novel approaches to distributed and parallel computing that are inspired by the mechanism and functioning of biological cells. More specifically, this thesis has focused on the concept of cellular distributed and parallel computing (in short, cellular computing) to solve the constraint satisfaction problem, the distributed maximal independent set selection problem, and the distributed greedy colouring problem.

Chapter 2 set out some basic background on cellular computing and the computational problems studied in the thesis.

Chapter 3 gave a theoretical polynomial-time solution to the constraint satisfaction problem based on neural-like P systems which are cellular computing models in the framework of membrane computing. A family of recognizer neural-like P systems was designed to solve the general CSP in polynomial time for the first time. By broadcasting symbol-impulses to neighbor cells and processing multisets or strings of symbol-impulses from neighbor cells, it was proved that neural-like P systems can in principle solve the CSP in polynomial time without using cell division or cell separation.

In Chapter 4, we implemented a bio-inspired algorithm on a bio-inspired model and designed a class of neural-like probabilistic P systems to solve one of the most fundamental distributed computational problems: the distributed maximal independent set selection problem. Our neural-like probabilistic P systems contain simple identical cells and use only one-bit messages to complete this distributed computational task. Moreover, the individual processors (cells) need no information about the size of the graph G, and use a simple and regular set of probability values (inverse pow-

ers of 2), which makes the class of neural-like probabilistic P system models proposed here easily implementable in very simple hardware.

In Chapter 5, we proposed and analysed a novel algorithmic scheme for the problem of distributed greedy colouring. This algorithmic scheme is derived from the approach used for the novel MIS selection algorithm inspired by the study of the neurological development of the fruit fly [AAB$^+$11, AABJ$^+$11]. Using this basic algorithmic scheme, we constructed two efficient distributed algorithms for greedy graph colouring. These new algorithms allow each processor to have minimal knowledge of the graph, and to send only simple messages corresponding to the set of colours to be used.

In Chapter 6, we gave an improved distributed MIS selection algorithm as well as an improved distributed greedy colouring algorithm, inspired by the intercellular signalling mechanism used for "fine-grained" pattern formation in many biological organisms. The improved algorithms incorporated for the first time an important feature of biological systems: adapting the probabilities used at each node based on local feedback from neighbouring nodes.

Our improved distributed MIS selection algorithm uses simple identical processors and one-bit messages, and its expected running time grows only logarithmically with the number of nodes, and is therefore much lower than the time required by a centralised sequential algorithm. Moreover, the individual processors need no information about the global properties of the network, such as its size, and do not need to identify which neighbours sent which message. These features make the algorithm useful for many applications, such as ad hoc sensor networks and wireless communication systems. Our improved distributed greedy colouring algorithm that matches state-of-the-art complexity works under similar very restrictive conditions where the simple identical processors have no information about the network and we allow each processor to broadcast only a single message to all neighbours at each time step representing a single desired colour value.

We also showed that our improved algorithms are highly robust, in the sense that they retain their good performance even when various features are changed. For example, the probabilities at each node do not need to increase and decrease by a precise factor – the analysis we have given in the thesis allows a wide range of different values for these factors, which may vary between nodes and over time. Similarly, the initial values for the probabilities at each node may be randomly chosen, and may vary from node to node, without any significant impact on performance (as long as they are bounded away from zero). This robustness is likely to be a key feature of the algorithms in any biological context.

Finally, Chapter 7 showed the actual performance of the distributed MIS selection algorithms and the distributed greedy colouring algorithms investigated in the thesis. The experiment results demonstrate their efficiency in practice on some important classes of graphs. Moreover, the performance of the algorithms on some standard graph colouring benchmarks has shown that the number

of colours used by the distributed greedy colouring algorithms is often optimal or near to the optimal value. Since these algorithms can be simulated easily and efficiently on a standard single processor, they offer a new heuristic approach to graph colouring problems where the aim is to find colourings with a small number of colours.

Computer science and biology have enjoyed a long and fruitful relationship for decades, and recently they have been converging [NBJ11]. As basic information processing units of life, biological cells have drawn broad interest from both computer scientists and biologists. In this thesis, we have shown that by drawing inspiration from biological cells we can construct very robust and efficient cellular computing approaches. Biological cells are inherently distributed and parallel and can work under a variety of harsh constraints and conditions, thus inspirations from biological cells especially fit to the area of distributed and parallel computing.

Thinking about biological processes computationally can often lead to better understanding of the biological systems and at the same time can improve the design of algorithms. The cellular computing approaches to the constraint satisfaction problem, the distributed maximal independent set selection problem and the distributed greedy colouring problem in this thesis thus further confirm the significance of the increasing convergence between systems biology and computational thinking [NBJ11].

8.2 Open problems

1. Although the brute-force algorithm taken by the neural-like P systems proposed in Chapter 3 is theoretically proved to be efficient for the CSP in terms of time, the number of objects generated by the family of recognizer neural-like P systems we describe is exponentially large, making the implementation of such a system rather impractical. In future research, can we build on more sophisticated techniques for solving the CSP [RBW06], in order to design neural-like P systems which avoid, where possible, the exponential explosion in the number of objects generated?

2. Implementing bio-inspired algorithms in computational models of biological systems is an interesting research topic which plays an important role on bridging between computational thinking and systems biology [NBJ11]. We have implemented the bio-inspired MIS selection algorithm by Afek et al. [AABJ$^+$11] successfully in a class of neural-like probabilistic P systems. As another example, Luca Cardelli and Attila Csikász-Nagy have successfully implemented an efficient approximate majority algorithm [AAE08] in chemical reaction networks to investigate the robustness and efficiency of the biological cell cycle switch [CCN12]. In future work, can we also implement our new MIS selection algorithm and our new greedy colouring algorithms on similar models of biological systems in a biologically realistic way?

3. We have proposed state-of-the-art algorithms for the fundamental distributed maximal independent set selection problem and the distributed greedy colouring problem. In future research, can we adapt the algorithms to solve other hard computational problems such as (distributed) constraint satisfaction problems?

4. In the distributed MIS selection problem, the MIS we choose is maximal but not necessarily maximum; Similarly, in the distributed greedy colouring problem, the colouring we obtain does not necessarily use the minimum number of colours. In future research, can we adapt our improved distributed MIS selection algorithm and distributed greedy colouring algorithm to make them better heuristic approaches to the corresponding **NP**-hard problems: the maximum independent set selection problem and the graph colouring problem with minimum number of colours?

5. The relationship between "fine-grained" pattern formation and the maximal independent set selection problem that we have investigated in the thesis is a strikingly successful example of the increasing convergence between systems biology and computational thinking [NBJ11]. "Fine-grained" pattern formation modelling is an intensively studied problem and there are many theoretical models of it [CMML96, Bra06, Arc13] in the literature. Almost all previous work assumes continuous models of Delta-Notch inter-cellular signalling, for example, Collier et al. use continuous increasing/decreasing functions to construct differential equations illustrating the levels of Notch and Delta activity in the cell [CMML96], and there's much less work on constructing discrete models which seem more suitable to be simulated on electronic computers. In future work, can we build a probabilistic discrete model for "fine-grained" pattern formation based on our improved distributed maximal independent set selection algorithm?

APPENDIX A

Source code

Table A.1: *Mathematica* code implementing the MIS selection algorithm in Table 4.1

```
initialise[g_] := (states = ConstantArray[0, VertexCount[g]];
                   beeps = ConstantArray[0, VertexCount[g]];
                   p = 1; k = 1;
                   adjacency = AdjacencyMatrix[g];)

update[0, 0] := If[RandomReal[] <= p, {2, 1}, {1, 0}];
update[0, _] := {3, 0};
update[1, _] := {0, 0};
update[2, 0] := {4, 1};
update[2, _] := {0, 0};
update[3, _] := {3, 0};
update[4, _] := {4, 0};

step := (beeps = beeps.adjacency;
        {states, beeps} = Transpose[MapThread[update, {states, beeps}]];)

chooseMIS[g_] := (initialise[g]; r = 0; countbeeps = 0;
                  While[Count[states, 0 | 1 | 2] >0, r++; step;
                  countbeeps += Count[states, 2]; step; p = p/2;
                  If[p < 1/2^k, p = 1; k++];])

chooseMIS[g];
Print[r, "time steps"];
```

Table A.2: *Mathematica* code implementing the MIS selection algorithm in Table 6.1

```
initialise1[g_] := (states = ConstantArray[0, VertexCount[g]];
                    beeps = ConstantArray[0, VertexCount[g]];
                    plist = ConstantArray[1, VertexCount[g]];
                    increase = 2; decrease = 1/increase;
                    adjacency = AdjacencyMatrix[g];)

update1[0, 0, p_] := If[RandomReal[] <= p, {2, 1, p}, {1, 0, p}];
update1[0, _, _] := {3, 0, 0};
update1[1, 0, p_] := {0, 0, Min[p*increase, 1]};
update1[1, _, p_] := {0, 0, p*decrease};
update1[2, 0, _] := {4, 1, 0};
update1[2, _, p_] := {0, 0, p*decrease};
update1[3, _, _] := {3, 0, 0};
update1[4, _, _] := {4, 0, 0};

step1 := (beeps = beeps.adjacency;
        {states, beeps, plist} = Transpose[MapThread[update1, {states, beeps, plist}]];)

chooseMIS1[g_] := (initialise1[g]; r = 0; countbeeps = 0;
                   While[Count[states, 0 | 1 | 2] > 0, r++; step1;
                   countbeeps += Count[states, 2]; step1;]);

chooseMIS1[g];
Print[r, "time steps"];
```

Table A.3: *Mathematica* code implementing the greedy colouring algorithms in Tables 5.2, 5.3

```
initialiseGC[g_] := (states = ConstantArray[0, VertexCount[g]];
                     beeps = ConstantArray[0, VertexCount[g]];
                     colours = ConstantArray[0, VertexCount[g]];
                     p = 1; k = 1;
                     adjacency = AdjacencyMatrix[g];)

firstavailable[forbidlist_] := First[Complement[Range[Length[forbidlist] + 1], forbidlist]]

updateGC[0, b_, c_] := If[RandomReal[] <= p, {1, newb = -firstavailable[b], newb}, {3, 0, 0}];
updateGC[1, b_, c_] := If[MemberQ[b, c], {0, 0, 0}, {2, -c, -c}];
updateGC[2, _, c_] := {2, c, c};
updateGC[3, b_, c_] := {0, 0, 0};

stepGC := (beeps = Inner[Times, adjacency, beeps, List];
          {states, beeps, colours} = Transpose[MapThread[updateGC, {states, beeps, colours}]];)

(*Algorithm in Table 5.2*)
chooseGC[g_] := (initialiseGC[g]; r = 0; countbeeps = 0;
                 While[Count[states, 0 | 1 | 3] > 0, r++; stepGC;
                 countbeeps += Count[states, 1]; stepGC; p = p/2;
                 If[p < 1/2^k, p = 1; k++];])

(*Algorithm in Table 5.3*)
chooseGCnew[g_] := (initialiseGC[g]; r = 0; countbeeps = 0;
                    While[Count[states, 0 | 1 | 3] > 0, r++; stepGC;
                    countbeeps += Count[states, 1]; stepGC; p = p/2;
                    If[p < 1/VertexCount[g], p = 1;];])

chooseGC[g]; χ = Max[colours];
Print[r, " time steps, ", χ, " colours, degree ", Max[VertexOutDegree[g]] ];

chooseGCnew[g]; χ = Max[colours];
Print[r, " time steps, ", χ, " colours, degree ", Max[VertexOutDegree[g]] ];
```

Table A.4: *Mathematica* code implementing the greedy colouring algorithm in Tables 6.2

```
initialiseGC1[g_] := (states = ConstantArray[0, VertexCount[g]];
                      beeps = ConstantArray[0, VertexCount[g]];
                      colours = ConstantArray[0, VertexCount[g]];
                      plist = ConstantArray[1, VertexCount[g]];
                      increase = 2; decrease = 1/increase;
                      adjacency = AdjacencyMatrix[g];)

firstavailable[forbidlist_] := First[Complement[Range[Length[forbidlist] + 1], forbidlist]]

updateGC1[0, b_, p_, c_] := If[RandomReal[] <= p, {1, newb = -firstavailable[b], p, newb}, {3, 0,
                              p, -firstavailable[b]}];
updateGC1[1, b_, p_, c_] := If[MemberQ[b, c], {0, 0, p*decrease, 0}, {2, -c, 0, -c}];
updateGC1[2, _, _, c_] := {2, c, 0, c};
updateGC1[3, b_, p_, c_] := If[MemberQ[b, c], {0, 0, p*decrease, 0}, {0, 0, Min[p*increase, 1], 0}]

stepGC1 := (beeps = Inner[Times, adjacency, beeps, List];
            {states, beeps, plist, colours} = Transpose[MapThread[updateGC1, {states, beeps, plist,
                                                  colours}]];)

chooseGC1[g_] := (initialiseGC1[g]; r = 0; countbeeps = 0;
                  While[Count[states, 0 | 1 | 3] > 0, r++; stepGC1;
                  countbeeps += Count[states, 1]; stepGC1;])

chooseGC1[g]; χ = Max[colours];
Print[r, " time steps, ", χ, " colours, degree ", Max[VertexOutDegree[g]] ];
```

Bibliography

[AAB+11] Y. Afek, N. Alon, O. Barad, E. Hornstein, N. Barkai, and Z. Bar-Joseph. A biological solution to a fundamental distributed computing problem. *Science*, 331(6014):183–185, 2011.

[AABJ+11] Y. Afek, N. Alon, Z. Bar-Joseph, A. Cornejo, B. Haeupler, and F. Kuhn. Beeping a maximal independent set. In *Proceedings of the 25th International Conference on Distributed Computing*, DISC'11, pages 32–50. Springer-Verlag, 2011.

[AAE08] D. Angluin, J. Aspnes, and D. Eisenstat. A simple population protocol for fast robust approximate majority. *Distributed Computing*, 21(2):87–102, 2008.

[ABI86] N. Alon, L. Babai, and A. Itai. A fast and simple randomized parallel algorithm for the maximal independent set problem. *Journal of Algorithms*, 7:567–583, 1986.

[AFL83] E. Arjomandi, M. J. Fischer, and N. A. Lynch. Efficiency of synchronous versus asynchronous distributed systems. *Journal of the ACM*, 30(3):449–456, 1983.

[All84] J. F. Allen. Towards a general theory of action and time. *Artificial Intelligence*, 23:123–154, July 1984.

[AMVP03] A. Alhazov, C. Martín-Vide, and L. Pan. Solving a PSPACE-complete problem by recognizing P systems with restricted active membranes. *Fundamenta Informaticae*, 58(2):67–77, April 2003.

[Arc13] M. Arcak. Pattern formation by lateral inhibition in large-scale networks of cells. *IEEE Transactions on Automatic Control*, 58(5):1250–1262, 2013.

[BE09] L. Barenboim and M. Elkin. Distributed $(\delta + 1)$-coloring in linear (in δ) time. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing*, STOC '09, pages 111–120, New York, NY, USA, 2009. ACM.

[BE10] L. Barenboim and M. Elkin. Sublogarithmic distributed MIS algorithm for sparse graphs using Nash-Williams decomposition. *Distributed Computing*, 22(5-6):363–379, 2010.

[BE11] L. Barenboim and M. Elkin. Deterministic distributed vertex coloring in polylogarithmic time. *Journal of the ACM*, 58(5):23:1–23:25, October 2011.

[BEPS12] L. Barenboim, M. Elkin, S. Pettie, and J. Schneider. The locality of distributed symmetry breaking. In *Proceedings of the 2012 IEEE 53rd Annual Symposium on Foundations of Computer Science*, FOCS '12, pages 321–330, Washington, DC, USA, 2012. IEEE Computer Society.

[BFM97] T. Back, D. B. Fogel, and Z. Michalewicz. *Handbook of Evolutionary Computation*. IOP Publishing Ltd., 1997.

[BHB11] O. Barad, E. Hornstein, and N. Barkai. Robust selection of sensory organ precursors by the Notch-Delta pathway. *Current Opinion in Cell Biology*, 23(6):663–667, 2011.

[BHS09] L. Bordeaux, Y. Hamadi, and H. Samulowitz. Experiments with massively parallel constraint solving. In *Proceedings of the 21st international jont conference on Artifical intelligence*, IJCAI'09, pages 443–448, San Francisco, CA, USA, 2009. Morgan Kaufmann Publishers Inc.

[Bis95] C. M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, Inc., 1995.

[BJL+82] J. E. Burns, P. Jackson, N. A. Lynch, M. J. Fischer, and G. L. Peterson. Data requirements for implementation of n-process mutual exclusion using a single shared variable. *Journal of the ACM*, 29(1):183–205, 1982.

[Bra06] S. J. Bray. Notch signalling: a simple pathway becomes complex. *Nature Reviews Molecular Cell Biology*, 7(9):678–689, 2006.

[BRHB10] O. Barad, D. Rosin, E. Hornstein, and N. Barkai. Error minimization in lateral inhibition circuits. *Science Signaling*, 3(129):ra51, 2010.

[CCM+11] M. Cardona, M. A. Colomer, A. Margalida, A. Palau, I. Pérez-Hurtado, M. J. Pérez-Jiménez, and D. Sanuy. A computational modeling for real ecosystems based on P systems. *Natural Computing*, 10(1):39–53, 2011.

[CCN12] L. Cardelli and A. Csikász-Nagy. The cell cycle switch computes approximate majority. *Scientific Reports*, 2, 2012.

[CdAPH+10] M. A. Colomer, M. A. M. del Amor, I. Pérez-Hurtado, M. J. Pérez-Jiménez, and A. Riscos-Núñez. A uniform framework for modeling based on P systems. In *2010 IEEE Fifth International Conference on Bio-Inspired Computing: Theories and Applications (BIC-TA)*, pages 616–621. IEEE, 2010.

[CFI+06] H. Chen, R. Freund, M. Ionescu, G. Păun, and M.-J. Pérez-Jiménez. On string languages generated by spiking neural P systems. In M. A. Gutiérrez-Naranjo, G. Păun, A. Riscos-Núñez, and F. J. Romero-Campero, editors, *Fourth Brainstorming Week on Membrane Computing, Sevilla, January 30 – February 3, 2006. Volume I*, pages 169–194. Fénix Editora, 2006.

[Cha79] I. Chakravarty. A generalized line and junction labeling scheme with application to scene analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1(2):202–205, February 1979.

[Cha87] P. Chaudhuri. Algorithms for some graph problems on a distributed computational model. *Information Sciences*, 43(3):205–228, 1987.

[CK10] A. Cornejo and F. Kuhn. Deploying wireless networks with beeps. In *Proceedings of the 24th International Conference on Distributed Computing*, DISC'10, pages 148–162, Berlin, Heidelberg, 2010. Springer-Verlag.

[CLRS09] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, 3rd edition, 2009.

[CMML96] J. R. Collier, N. A. Monk, P. K. Maini, and J. H. Lewis. Pattern formation by lateral inhibition with feedback: a mathematical model of delta-notch intercellular signalling. *Journal of Theoretical Biology*, 183(4):429–446, 1996.

[CPPJ06] G. Ciobanu, G. Păun, and M. J. Pérez-Jiménez. *Applications of Membrane Computing*. Springer, 2006.

[Cri88] J. M. Crichlow. *An Introduction to Distributed and Parallel Computing*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1988.

[Dec87] R. Dechter. A constraint-network approach to truth maintenance. Technical Report R-870009, Cognitive Systems Laboratory, Computer Science Department, University of California at Los Angeles, 1987.

[DPGNPJRNn08] D. Díaz-Pernil, M. A. Gutiérrez-Naranjo, M. J. Pérez-Jiménez, and A. Riscos-Núñez. A uniform family of tissue P systems with cell division solving 3-COL in a linear time. *Theoretical Computer Science*, 404(1-2):76–87, September 2008.

[DPGOGN$^+$09] D. Díaz-Pernil, P. Gallego-Ortiz, M. A. Gutiérrez-Naranjo, M. J. Pérez-Jiménez, and A. Riscos-Núñez. Cell-like versus tissue-like P systems by means of Sevilla carpets. In R. Gutiérrez-Escudero, M. Á. Gutiérrez-Naranjo, G. Paun, I. Pérez-Hurtado, and A. Riscos-Núñez, editors, *Seventh Brainstorming Week on Membrane Computing, Sevilla, February, 2009. Volume I*, pages 109–121. Fénix Editora, 2009.

[DR90] V. Dhar and N. Ranganathan. Integer programming vs. expert systems: an experimental comparison. *Communications of the ACM*, 33:323–336, March 1990.

[Eng05] A. P. Engelbrecht. *Fundamentals of computational swarm intelligence*, volume 1. Wiley Chichester, 2005.

[EW13] Y. Emek and R. Wattenhofer. Stone age distributed computing. In *Proceedings of the 2013 ACM Symposium on Principles of Distributed Computing*, PODC '13, pages 137–146, New York, NY, USA, 2013. ACM.

[FGPJ14] P. Frisco, M. Gheorghe, and M. J. Pérez-Jiménez. *Applications of Membrane Computing in Systems and Synthetic Biology*. Springer, 2014.

[FLP85] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382, 1985.

[GHR95] R. Greenlaw, H. J. Hoover, and W. L. Ruzzo, editors. *Limits to Parallel Computation: P-completeness Theory*. Oxford University Press, Inc., New York, NY, USA, 1995.

[GKAT12] K. G. Guruharsha, M. W. Kankel, and S. Artavanis-Tsakonas. The Notch signalling system: recent insights into the complexity of a conserved pathway. *Nature Reviews Genetics*, 2012.

[GKK+09] C. Gavoille, R. Klasing, A. Kosowski, Ł. Kuszner, and A. Navarra. On the complexity of distributed graph coloring with local minimality constraints. *Networks*, 54(1):12–19, 2009.

[GLGT03] A. H. Gebremedhin, I. G. Lassous, J. Gustedt, and J. A. Telle. Graph coloring on coarse grained multicomputers. *Discrete Applied Mathematics*, 131(1):179–198, September 2003.

[GM75] G. Grimmett and C. McDiarmid. On colouring random graphs. *Mathematical Proceedings of the Cambridge Philosophical Society*, 77:313–324, 2 1975.

[GNPJRC05] M. A. Gutiérrez-Naranjo, M. J. Pérez-Jiménez, and F. Romero-Campero. A linear solution of subset sum problem by using membrane creation. In J. Mira and J. R. Alvarez, editors, *International Work-Conference on the Interplay Between Natural and Artificial Computation, IWINAC'05*, Lecture Notes in Computer Science 3561, pages 258–267. Springer, 2005.

[GPS88] A. V. Goldberg, S. A. Plotkin, and G. E. Shannon. Parallel symmetry-breaking in sparse graphs. *SIAM Journal on Discrete Mathematics*, 1(4):434–446, October 1988.

[Gru39] P. M. Grundy. Mathematics and games. *Eureka*, 2:6–8, 1939.

[HJS02] S. T. Hedetniemi, D. P. Jacobs, and P. K. Srimani. Fault tolerant distributed coloring algorithms that stabilize in linear time. In *Proceedings of the 16th International Parallel and Distributed Processing Symposium*, IPDPS '02, pages 146–150, Washington, DC, USA, 2002. IEEE Computer Society.

[HJS03] S. T. Hedetniemi, D. P. Jacobs, and P. K. Srimani. Linear time self-stabilizing colorings. *Information Processing Letters*, 87(5):251–255, September 2003.

[HKKN04] J. Hansen, M. Kubale, Ł. Kuszner, and A. Nadolski. Distributed largest-first algorithm for graph coloring. In M. Danelutto, M. Vanneschi, and D. Laforenza, editors, *Euro-Par 2004 Parallel Processing*, volume 3149 of *Lecture Notes in Computer Science*, pages 804–811. Springer Berlin Heidelberg, 2004.

[HM90] J. Y. Halpern and Y. Moses. Knowledge and common knowledge in a distributed environment. *Journal of the ACM*, 37(3):549–587, 1990.

[ILP+10] T.-O. Ishdorj, A. Leporati, L. Pan, X. Zeng, and X. Zhang. Deterministic solutions to QSAT and Q3SAT by spiking neural P systems with pre-computed resources. *Theoretical Computer Science*, 411(25):2345–2358, May 2010.

[IMVP01] M. Ito, C. Martn-Vide, and G. Păun. A characterization of parikh sets of ET0L languages in terms of P systems. In *Words, Semigroups, and Transductions'01*, pages 239–253, 2001.

[IMVPP03] M. Ionescu, C. Martín-Vide, A. Păun, and G. Păun. Unexpected universality results for three classes of P systems with symport/antiport. *Natural Computing*, 2(4):337–348, 2003.

[IPY06] M. Ionescu, G. Păun, and T. Yokomori. Spiking neural P systems. *Fundamenta Informaticae*, 71(2-3):279–308, 2006.

[IR90] A. Itai and M. Rodeh. Symmetry breaking in distributed networks. *Information and Computation*, 88(1):60–87, 1990.

[JBAM98] T. L. Jacobsen, K. Brennan, A. M. Arias, and M. A. Muskavitch. Cis-interactions between Delta and Notch modulate neurogenic signalling in Drosophila. *Development*, 125(22):4531–4540, 1998.

[JSX14] P. Jeavons, A. Scott, and L. Xu. Feedback from nature: randomised distributed algorithms for maximal independent set selection and greedy colouring. *in submission*, 2014.

[Kar72] R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972.

[Kep11] J. O. Kephart. Learning from nature. *Science*, 331(6018):682–683, 2011.

[KMNW05] F. Kuhn, T. Moscibroda, T. Nieberg, and R. Wattenhofer. Fast deterministic distributed maximal independent set computation on growth-bounded graphs. In P. Fraigniaud, editor, *Distributed Computing: 19th International Conference, DISC 2005*, volume 3724 of *Lecture Notes in Computer Science*, pages 273–283. Springer, November 2005.

[KMW04] F. Kuhn, T. Moscibroda, and R. Wattenhofer. What cannot be computed locally! In *Proceedings of the 23rd Annual ACM Symposium on Principles of Distributed Computing*, PODC '04, pages 300–309, New York, NY, USA, 2004. ACM.

[KMW06] F. Kuhn, T. Moscibroda, and R. Wattenhofer. The price of being near-sighted. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithm*, SODA '06, pages 980–989, New York, NY, USA, 2006.

[Kro11] K. L. Kroeker. Biology-inspired networking. *Communications of the ACM*, 54:11–13, June 2011.

[Kuh09] F. Kuhn. Weak graph colorings: distributed algorithms and applications. In *Proceedings of the 21st Annual Symposium on Parallelism in Algorithms and Architectures*, SPAA '09, pages 138–144, New York, NY, USA, 2009. ACM.

[KW85] R. M. Karp and A. Wigderson. A fast parallel algorithm for the maximal independent set problem. *Journal of the ACM*, 32(4):762–773, 1985.

[KW06] F. Kuhn and R. Wattenhofer. On the complexity of distributed graph coloring. In *Proceedings of the 25th Annual ACM Symposium on Principles of Distributed Computing*, PODC '06, pages 7–15, New York, NY, USA, 2006. ACM.

[Lin86] N. Linial. Legal coloring of graphs. *Combinatorica*, 6(1):49–54, 1986.

[Lin87] N. Linial. Distributive graph algorithms – global solutions from local data. In *Proceedings of the 28th Annual Symposium on Foundations of Computer Science*, SFCS '87, pages 331–335, Washington, DC, USA, 1987. IEEE Computer Society.

[Lin92] N. Linial. Locality in distributed graph algorithms. *SIAM Journal on Computing*, 21(1):193–201, February 1992.

[LK99] L. F. Landweber and L. Kari. The evolution of cellular computing: Natures solution to a computational problem. *Biosystems*, 52(1):3–13, 1999.

[Lub85] M. Luby. A simple parallel algorithm for the maximal independent set problem. In *Proceedings of the seventeenth annual ACM symposium on theory of computing*, STOC '85, pages 1–10, New York, NY, USA, 1985.

[Lub86] M. Luby. A simple parallel algorithm for the maximal independent set problem. *SIAM Journal on Computing*, 15(4):1036–1053, 1986.

[LV02] J. Larrosa and G. Valiente. Constraint satisfaction algorithms for graph pattern matching. *Mathematical Structures in Computer Science*, 12(4):403–422, August 2002.

[LW12] C. Lenzen and R. Wattenhofer. Distributed algorithms for sensor networks. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 370(1958):11–26, January 2012.

[Lyn96] N. A. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1996.

[LZFM07] A. Leporati, C. Zandron, C. Ferretti, and G. Mauri. Solving numerical NP-complete problems with spiking neural P systems. In *Proceedings of the 8th international conference on membrane computing*, WMC'07, pages 336–352, Berlin, Heidelberg, 2007. Springer-Verlag.

[MC09] M. Matsuda and A. B. Chitnis. Interaction with Notch determines endocytosis of specific Delta ligands in zebrafish neural tissue. *Development*, 136(2):197–206, 2009.

[MMG+07] A. Mislove, M. Marcon, K. P. Gummadi, P. Druschel, and B. Bhattacharjee. Measurement and analysis of online social networks. In *Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement*, IMC '07, pages 29–42, New York, NY, USA, 2007. ACM.

[MP43] W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, 5(4):115–133, 1943.

[MP12] V. Maan and G. N. Purohit. A distributed approach for frequency allocation using graph coloring in mobile networks. *International Journal of Computer Applications*, 58(6):9–13, November 2012. Published by Foundation of Computer Science, New York, USA.

[MRSDZ10] Y. Métivier, J. M. Robson, N. Saheb-Djahromi, and A. Zemmari. About randomised distributed graph colouring and graph partition algorithms. *Information and Computation*, 208(11):1296–1304, 2010.

[MRSDZ11] Y. Métivier, J. M. Robson, N. Saheb-Djahromi, and A. Zemmari. An optimal bit complexity randomized distributed MIS algorithm. *Distributed Computing*, 23(5-6):331–340, 2011.

[MVPPRP03] C. Martín-Vide, G. Păun, J. Pazos, and A. Rodríguez-Patón. Tissue P systems. *Theoretical Computer Science*, 296(2):295–326, 2003.

[MW05] T. Moscibroda and R. Wattenhofer. Maximal independent sets in radio networks. In *Proceedings of the 24th Annual ACM Symposium on Principles of Distributed Computing*, PODC '05, pages 148–157, New York, NY, USA, 2005. ACM.

[NBJ11] S. Navlakha and Z. Bar-Joseph. Algorithms in nature: the convergence of systems biology and computational thinking. *Molecular systems biology*, 7(1), 2011.

[Nic12] R. Nicolescu. Parallel and distributed algorithms in P systems. In M. Gheorghe, G. Păun, G. Rozenberg, A. Salomaa, and S. Verlan, editors, *12th International Conference on Membrane Computing*, volume 7184 of *Lecture Notes in Computer Science*, pages 35–50. Springer, 2012.

[NSW11] J. Ni, R. Srikant, and X. Wu. Coloring spatial point processes with applications to peer discovery in large wireless networks. *IEEE/ACM Transactions on Networking*, 19(2):575 –588, April 2011.

[NW11] R. Nicolescu and H. Wu. BFS solution for disjoint paths in P systems. In *Proceedings of the 10th international conference on unconventional computation*, UC'11, pages 164–176, Berlin, Heidelberg, 2011. Springer-Verlag.

[NZS+06] R. Nagpal, F. Zambonelli, E. G. Sirer, H. Chaouchi, and M. Smirnov. Interdisciplinary research: Roles for self-organization. *IEEE Intelligent Systems*, 21(2):50–58, 2006.

[OJ99] Öjvind Johansson. Simple distributed $\Delta + 1$-coloring of graphs. *Information Processing Letters*, 70(5):229 – 232, 1999.

[Pap03] C. H. Papadimitriou. *Computational Complexity*. John Wiley and Sons Ltd., 2003.

[Pel00] D. Peleg. *Distributed Computing: A Locality-sensitive Approach*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000.

[PI04] L. Pan and T. O. Ishdorj. P systems with active membranes and separation rules. *Journal of Universal Computer Science*, 10(5):630–649, May 2004.

[PJ09] M. J. Pérez-Jiménez. Computational complexity in P systems. *Scholarpedia*, 4(11):9290, 2009.

[PJ10] M. J. Pérez-Jiménez. A computational complexity theory in membrane computing. *Lecture Notes in Computer Science*, 5957:125–148, 01/2010 2010. (invited talk). Membrane Computing, 10th International Workshop, WMC 2009, Curtea de Arges, Romania, August 24-27, 2009, Revised Selected and Invited Papers.

[PJJC06] M. J. Pérez-Jiménez, A. R. Jiménez, and F. S. Caparrini. A polynomial complexity class in P systems using membrane division. *Journal of Automata, Languages and Combinatorics*, 11:423–434, January 2006.

[PL96] T. Park and C. Y. Lee. Application of the graph coloring algorithm to the frequency assignment problem. *Journal of the Operations Research Society of Japan-Keiei Kagaku*, 39(2):258–265, 1996.

[PPJ10] L. Pan and M. J. Pérez-Jiménez. Computational complexity of tissue-like P systems. *Journal of Complexity*, 26(3):296–315, June 2010.

[PPJRN08] G. Păun, M. J. Pérez-Jiménez, and A. Riscos-Núñez. Tissue P systems with cell division. *International Journal of Computers, Communications & Control*, 3(3):295–303, 2008.

[PPPJ11] L. Pan, G. Păun, and M. J. Pérez-Jiménez. Spiking neural P systems with neuron division and budding. *SCIENCE CHINA Information Sciences*, 54(8):1596–1607, 2011.

[PR01] A. Panconesi and R. Rizzi. Some simple distributed algorithms for sparse networks. *Distributed Computing*, 14(2):97–100, April 2001.

[PRS97] R. Prakash, M. Raynal, and M. Singhal. An adaptive causal ordering algorithm suited to mobile computing environments. *Journal of Parallel and Distributed Computing*, 41(2):190–204, 1997.

[PRS10] G. Păun, G. Rozenberg, and A. Salomaa, editors. *The Oxford Handbook of Membrane Computing*. Oxford University Press, Inc., New York, NY, USA, 2010.

[PS96] A. Panconesi and A. Srinivasan. On the complexity of distributed network decomposition. *Journal of Algorithms*, 20(2):356–374, 1996.

[Pŏŏ] G. Păun. Computing with membranes: Attacking NP-complete problems. In I. Antoniou, C. Calude, and M. Dinneen, editors, *Unconventional Models of*

Computation, pages 94–115, London, February 2000. Springer-Verlag. Invited paper.

[PWH12] L. Pan, J. Wang, and H. J. Hoogeboom. Spiking neural P systems with astrocytes. *Neural Computation*, 24(3):805–825, 2012.

[PZZ11] L. Pan, X. Zeng, and X. Zhang. Time-free spiking neural P systems. *Neural Computation*, 23(5):1320–1342, 2011.

[RBW06] F. Rossi, P. v. Beek, and T. Walsh. *Handbook of Constraint Programming (Foundations of Artificial Intelligence)*. Elsevier Science Inc., New York, NY, USA, 2006.

[RCPJ08] F. J. Romero-Campero and M. J. Pérez-Jiménez. A model of the quorum sensing system in vibrio fischeri using P systems. *Artificial Life*, 14(1):95–109, January 2008.

[Sip99] M. Sipper. The emergence of cellular computing. *Computer*, 32(7):18–26, July 1999.

[SJX13] A. Scott, P. Jeavons, and L. Xu. Feedback from nature: an optimal distributed algorithm for maximal independent set selection. In *Proceedings of the 2013 ACM Symposium on Principles of Distributed Computing*, PODC '13, pages 147–156, New York, NY, USA, 2013. ACM.

[SLL+10] D. Sprinzak, A. Lakhanpal, L. LeBon, L. A. Santat, M. E. Fontes, G. A. Anderson, J. Garcia-Ojalvo, and M. B. Elowitz. Cis-interactions between Notch and Delta generate mutually exclusive signalling states. *Nature*, 465(7294):86–90, 2010.

[SW08] J. Schneider and R. Wattenhofer. A log-star distributed maximal independent set algorithm for growth-bounded graphs. In *Proceedings of the 27th ACM Symposium on Principles of Distributed Computing*, PODC '08, pages 35–44, New York, NY, USA, 2008. ACM.

[SW10] J. Schneider and R. Wattenhofer. A new technique for distributed symmetry breaking. In *Proceedings of the 29th ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, PODC '10, pages 257–266, New York, NY, USA, 2010. ACM.

[TA07] K. Tumer and A. Agogino. Distributed agent-based air traffic flow management. In *Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS '07, pages 255:1–255:8, New York, NY, USA, 2007. ACM.

[Upf84] E. Upfal. Efficient schemes for parallel communication. *Journal of the ACM*, 31(3):507–517, 1984.

[Wat05] R. J. Waters. *Graph Colouring and Frequency Assignment*. PhD thesis, 2005.

[Wat07] R. Wattenhofer. `http://dcg.ethz.ch/lectures/fs08/distcomp/lecture/chapter4.pdf`, 2007.

[WHP11] J. Wang, H. Hoogeboom, and L. Pan. Spiking neural P systems with neuron division. In M. Gheorghe, T. Hinze, G. Păun, G. Rozenberg, and A. Salomaa, editors, *Membrane Computing*, volume 6501 of *Lecture Notes in Computer Science*, pages 361–376. Springer Berlin Heidelberg, 2011.

[WMSZ09] S. Wang, Z. Miao, X. Shi, and Z. Zhang. Solving 3-coloring problem by tissue P systems with cell separation. In *Fourth International Conference on Bio-Inspired Computing, BIC-TA '09*, pages 1–6, October 2009.

[Wol94] S. Wolfram. *Cellular Automata and Complexity: Collected Papers*, volume 1. Addison-Wesley Reading, 1994.

[XJ13] L. Xu and P. Jeavons. Simple neural-like P systems for maximal independent set selection. *Neural computation*, 25(6):1642–1659, 2013.

[XJ14] L. Xu and P. Jeavons. Patterns from nature: distributed greedy colouring with simple messages and minimal graph knowledge. *in submission*, 2014.

[XLZ10] L. Xu, C. Lu, and Z. Zhang. Membrane systems with peripheral proteins using cell division. *Journal of Computational and Theoretical Nanoscience*, 7(11):2355–2359, 2010.

[Xu12] L. Xu. Modelling to contain pandemic influenza A (H1N1) with stochastic membrane systems: A work-in-progress paper. In J. Suzuki and T. Nakano, editors, *Bio-Inspired Models of Network, Information, and Computing Systems*, volume 87 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 74–81. Springer Berlin Heidelberg, 2012.

[XZJ13] L. Xu, X. Zeng, and P. Jeavons. A polynomial-time solution to constraint satisfaction problems by neural-like P systems. *International Journal of Unconventional Computing*, 9(5-6):465–481, 2013.

[Zab88] R. Zabih. A rearrangement search strategy for determining propositional satisfiability. In *Proceedings of the National Conference on Artificial Intelligence*, pages 155–160, 1988.

[Zuc06] D. Zuckerman. Linear degree extractors and the inapproximability of max clique and chromatic number. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing*, STOC '06, pages 681–690, New York, NY, USA, 2006. ACM.