THE UNIVERSITY of York

Distributed asynchronous heuristics for graph colouring

James Evans

BT Group plc
Mobility Research
Adastral Park
Martlesham Heath
Suffolk
IP5 3RE
United Kingdom

University of York
Department of Mathematics
Heslington
York
YO10 5DD
United Kingdom

Supervisors: Dr. K. M. Briggs
Dr. Z. Coelho

Dissertation submitted for the MSc in Mathematics with Modern Applications, Department of Mathematics, University of York, UK on 20th August 2010

Contents

Chapter 1

Introduction

Motivation

Wireless-enabled mobile devices are increasingly becoming more common and the networks in which they operate are becoming larger and more complex. Wireless communication has many benefits in modern-day life, but has the problem of interference and reduced performance when two devices are trying to access the same channel simultaneously. This can be seen as a resource allocation problem where a limited number of costly channels are in high demand. Currently global systems are used to tell devices which channel and transmission power level to use. To optimize the networks using globally techniques incurs large costs which will only increase with growing network sizes. Therefore, by programming each device to behave autonomously, the need for the global authority can be removed, with the optimization of the network being obtained using purely local knowledge. This report will look at ways of achieving this using the framework of distributed [15] [8] asynchronous heuristics for solving graph colouring problems.

Concepts

The three main areas that require consideration in this report are:

- conflict resolution
- distributed nature
- asynchronicity

Conflicts occur when two parties are after the same scarce resource, so in a wireless network this is when two neighbouring devices are both using the same channel and

ways to resolve this need to be considered. Distributed nature is the removal of a central authority which communicates information to all devices, meaning that devices can only make decisions based on knowledge they gather locally. Asynchronicity requires that devices activate to try and optimize the network at independent points in time. All of these concepts are seen in different walks of life, with examples for each as follows:

1. Commonly conflicts occur between two neighbouring countries where both claim ownership of an area of land, usually containing a valuable commodity. The result is usually war between the two, with years of unrest occurring afterwards. This is obviously not good in networks, meaning devices will be required to exhibit altruistic behaviour towards each other. This means a device may have to perform sub-optimally to allow for a better resolution of the whole network.

2. Species often use skin colour as a way of recognition when searching for a compatible mate. Conflicts arise when a species branches out from its current location and comes across another species using the same colour. Usually the species will attempt to change colour by obtaining pigments from the local environment enabling the conflict to be resolved. If this is successful then the colour change will be passed on through breeding and the species will be able to survive in the new location.

3. Originally most communities spoke only in their own local dialect. Then as people migrated to the larger communities they were forced to change their dialect due to the social pressures placed upon them. As this continued over time the dialects in the smaller communities grew obsolete, eventually resulting in one overall standard language developing.

4. To achieve the distributed nature, each device will be programmed to follow a set of rules which cause different reactions as their local environment changes. An example of this in nature can be seen in ant colonies. A study by Dorigo et al [10] shows how ants with only local knowledge are able to solve global problems to an almost optimal standard. They look at finding the shortest path between two points, which for ants is between their home and feeding grounds. Individual ants move randomly, but leave behind a pheromone trail. When the next ant comes along and senses this trail, it has an increased probability of taking the same route. For large colonies, this trail builds up most strongly along the

shortest route between two points, allowing all ants in the colony to successfully transverse between the two points with no global knowledge.

5. In nature, asynchronous hatching occurs when a group of eggs hatch over a few days rather than within a few hours of each other. This is induced by birds beginning the incubation of each egg as soon as it is laid rather than only starting once all of the eggs are laid. Asynchronous hatching is seen as a common parental strategy to bring up the largest family of offspring when the food resources available at the time of hatching are unknown by the bird when the eggs are initially laid [1].

6. Asynchronous behaviour is also seen in the flowering pattern of gorse bushes. During the colder winter months when the abundance of pollinating insects around is low, only a small number of bushes come into bloom. In the local area around each flowering gorse bush, none of its neighbours are in bloom at the same time. However, which of the gorse bushes flower during this period appears to be random, with no apparent pattern occurring year on year.

Overview

This report begins by considering real-life problems which can be modeled using graph colouring techniques. Optimization techniques for systems are then investigated and previous attempts to create distributed heuristics using synchronous time are considered. The framework of the distributed asynchronous heuristics will then be introduced and their workings detailed. The heuristics will be broken down into three classes and simulation results for these will be discussed in Chapters 4 to 6. These simulations will be carried out on random graphs of different structure with the performance of heuristics being judged. Finally these heuristics will be applied to a real-life example from the village of Martlesham Heath.

Acknowledgments

I would like to thank BT for allowing me the opportunity to work within their organization while carrying out this project and Zaq Coehlo for the support and guidance he has given me throughout the Masters course. Finally, my thanks to Keith Briggs for all the support and time he has given to me while carrying out this project.

Chapter 2

Background

2.1 Communications

In communications *orthogonality* is used to describe a scheme where all receiving devices can pick up their desired signal without interference from others. There are numerous ways highlighted below in which this can be achieved and all can be replicated using graph colouring.

Frequency-division multiple access - FDMA

FDMA works by splitting a single channel into separate blocks which do not overlap. Every device then uses a different block on which to communicate, so there is no interference and the system is orthogonal. For graph colouring, every one of these blocks can be seen as a separate colour. To work FDMA requires costly filters, but has the advantage of not been affected by timing variations as each block is always open to broadcast across.

Time-division multiple access - TDMA

TDMA uses one channel and allocates each device a small time frame in which it can communicate. These time frames form a sequence which is then continuously repeated. Now each time frame in the sequence can be seen as a colour, however, for TDMA to work a global processor is required to allocate and regulate time slots so locality is lost. With every device using the same channel costs are kept down but, as devices move around, their time delay between accessing the signal constantly changes. This can lead to devices broadcasting at the same time and orthogonality being lost.

Due to this, buffer gaps are implemented between time frames and this means the channel is used less efficiently.

Long term evolution - LTE

LTE is in the process of being introduced in communications at the moment and provides a combination of FDMA and TDMA. This means a channel is divided into blocks and then devices are given a small time frame in which they can use a set block. This means that a device can jump around blocks every time frame and hence higher throughput. Every block per time frame can be assigned a colour.

Code-division multiple access - CDMA

CDMA operates using one channel and a device can communicate whenever it likes. To achieve orthogonality, when a device sends out a message it encodes it. Then only the intended recipients have the correct code to decode the message, preventing interference from other messages. Now each colour represents a different orthogonal code.

CDMA can also be carried out asynchronously and this is done using a 'pseudo-noise' sequence which is assigned to each code. The main advantage of this is it allows the channel to be used most efficiently when compared to FDMA, TDMA and synchronous CDMA.

2.2 Simulated annealing

Simulated annealing is a global optimization technique whereby a system slowly changes its state in an attempt to improve the systems overall performance. The name comes from chemistry where metals are heated and cooled under controlled conditions to increase crystal size and reduce defects. This allows properties such as strength to be improved. A system starts with a given energy level, e, and then a small change is made, giving a new energy level of e'. Optimal solutions for the system are achieved at lower energy levels, so if $e' < e$ then the change is accepted. However, even if $e' > e$ then the change is accepted with a given probability defined by $P(e, e', T)$, where T is the current temperature of the system. This is to allow the system to move 'uphill' and prevent it from becoming stuck in a sub-optimal state. Generally $P(e, e', T)$ decreases as T decreases, so 'uphill' changes are accepted less frequently over time. This is because

T starts at a given value and then decays to zero value, whereby the system becomes frozen in a final, hopefully optimal, state.

Several of the ideas simulated annealing uses could prove useful when developing the distributed asynchronous heuristics. However, due to the global information required for the technique to operate successfully, the technique is not as useful for wireless networks. There are also several parameters which need to be defined when the system is initialized.

- What value of T should the system by initially heated to?
- How quickly should T decay?
- How is $P(e, e', T)$ defined?

Another flaw with simulated annealing for wireless networks is that once the system has been frozen new devices may not enter the network without the system having to be reheated, which again requires global communication to all devices. Ideally, once a system has been initialized it should run indefinitely with the structure of the network always changing as devices enter/leave and move around with their neighbours always changing.

2.3 Satisfiability problem

Satisfiability problems (SATs) are logic problems where variables are assigned Boolean values, true or false, to try and solve a given formula. The formula is constructed using the logic operators AND, OR and NOT ($\wedge, \vee, \neg$ respectively) and *literals*, which are either a variable or its negation. An example formula in conjunctive normal form (CNF) is shown in equation 2.1.

$$f(\underline{x}) = (x_0 \vee \neg x_1) \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge (x_4 \vee \neg x_1) \tag{2.1}$$

In CNF the formula consists of *clauses* built using the literals and the OR operator, and these are joined together with the AND operator. The formula is then only satisfied if every clause is satisfied.

There are many algorithms for solving SATs, some more effective than others. One simple way is brute force, where every possible combination of literal values is tried until a solution is reached. This will always find a solution, if one exists, but is very computational heavy even for small problems. Pumphrey [13] studied the sum-product

algorithm with 3-SAT, where each clause has precisely 3 literals. He introduced message passing between variables to try and improve the performance of the solver. His method works by using belief networks which use conditional probabilities based on the values the variables can take. He showed that this technique improved the solver in terms of returning optimal solutions in a given time frame.

Determining an upper bound for the chromatic number, χ (section 2.6.5), of a graph is one such problem which can be formulated as a SAT. Ayanegui and Chavez-Aragon [7] show how this can be done and investigated ways of returning tight upper bounds for χ. They showed that even for small graphs the reduction of the problem to CNF creates many clauses. For example, the complete graph with 7 nodes requires 308 clauses.

2.4 Previous work

Using distributed algorithms for graph colouring has previously been considered by Fitzpatrick and Meertens [12]. For their simulations they used synchronous time models, meaning every device in the network attempts to resolve conflicts at the same time. They showed that by introducing small activation probabilities for a device at each time step, better results could be obtained for a network. This is because it reduced the probability of *coherence*, being when two neighbouring devices both change to the same channel on the same time step, thus not reducing the number of conflicts.

Two areas in which this can be expanded are considered in this report. The first is introducing asynchronous time which will eliminate the problem of coherence within the network and the second is the 'intelligence' of each device. To achieve this, devices will be programmed to react in different ways depending on the current state of their local environment.

2.5 Definitions

Distributed

A *distributed* network is one where each device has its own processor and is able to make decisions for itself, known as *autonomous behaviour*. This means there is no central system which has knowledge of the whole network, with each device only having information about its neighbours.

Asynchronous

In an *asynchronous* network all the devices activate independently from each other, with no pattern or rules for when this occurs. This can achieved by letting each device have its own internal clock and then activating itself at random times.

Algorithm

An *algorithm* is a set of rules and instructions which when followed solve a given problem. It can be mathematically proved that these rules always provide the optimal solution.

Heuristic

A *heuristic* provides a set of rules which act as a guideline for solving a problem. They are based upon previous experience and are designed to generally guide the problem towards an acceptable resolution. It cannot be mathematically proved that these rules always find the optimal solution.

2.6 Graph theory

To tackle the problem of reducing interference in wireless networks, mathematical concepts from graph theory are required. A communications network can be represented by either an interference graph or a communication graph. In the interference case all edges in the graph show where possible conflicts and loss of performance between two devices can occur. In a communication graph the edges represent which devices can talk and thus share information. This report will consider interference graphs and use unweighted, undirected graphs to represent a network as set out below.

1. Each node in the graph represents a different device in the network, where n is the number of nodes in the graph.

2. An edge indicates where interference can occur between the two nodes it connects. Using unweighted, undirected graphs means each edge has no value and the two nodes it connects can both interfere with each other. This will be denoted $i \leftrightarrow j$, $i, j \in 0, 1, ..., n-1$.

3. The colour of each node, c_i, $i \in 0, 1, ..., n-1$, represents the channel it is currently operating on. This will range from 0 to $C-1$, where C is the maximum number of colours a graph is allowed.

4. A conflict occurs when two connected nodes have the same colour, so c_i=c_j for $i \leftrightarrow j$.

In wireless networks devices are capable of moving around freely and turning on and off, meaning the network structure is constantly changing with time. The channel usage of each device also changes as the network is optimized. Graph and network processes are introduced to hold information as follows.

Graph process

A *graph process* contains information about the topological structure of the related network at time t. It contains information about which nodes are currently in the network and where conflicts can occur, Ngraph(t) and Egraph(t) respectively, which are defined as

$$\begin{aligned} \text{Ngraph}(t) &= \{0, 1, \ldots, n(t)-1\} \\ \text{Egraph}(t) &= \{(0, k_0), (0, k_1), \ldots, (1, \ldots), \ldots)\} \text{ st } k_i \in \text{Ngraph}(t) \end{aligned}$$

where $n(t)$ is the number of nodes in the network at time t.

Network process

A *network process* contains global information about the current state of the system. At time t it is defined as

$$\text{network}(t) = (\text{graph}(t), \{c_i \text{ for } i \in \text{Ngraph(t)}\})$$

where graph(t) is a graph process and $c_i \in \{0, \ldots, C-1\}\ \forall\ i$.

Throughout this report several different ways of constructing random graphs will be considered, working from the theoretical towards more realistic representations of real-life networks and the following will be used to help define these.

Definition 1 *The degree of a node,* $\deg(i)$*, is the number of edges incident on that node*

Definition 2 *For a graph of n nodes, the mean degree, denoted* $\overline{d}$*, is* $\sum_{i=0}^{n-1} \deg(i)/n$

Definition 3 *The number of conflicts a node has,* $\text{conf}(i)$*, is the total number of neighbours which have the same colour*

For all of the graphs mentioned, the values stated will apply at time $t = 0$, so when the network is first initialized.

2.6.1 Erdős-Rényi model

The Erdős-Rényi random graph was first defined by Paul Erdős and Alfréd Rényi in a paper written in 1959 [11]. This model has standard notation $G(n, p)$ where n is the number of nodes in the graph and p is the probability of an edge occurring between any two nodes. The degree of each node is based upon a binomial distribution, where the following hold for any $G(n, p)$ graph:

- mean number of edges: $\overline{m} = \binom{n}{2}p = n(n-1)p/2$
- mean degree: $\overline{d} = (n-1)p$

Throughout this report the Erdős-Rényi model will be referred to by either $G(n, p)$, $G[n, \overline{m}]$ or $G\{n, \overline{d}\}$, with $\overline{m}$ and $\overline{d}$ defined in the equations above.

One of the main advantages of studying $G(n, p)$ graphs is that they are dimensionless as there is no set position for each node. However, the topological makeup of the graph is less likely to resemble a wireless network.

2.6.2 Geometric random graph

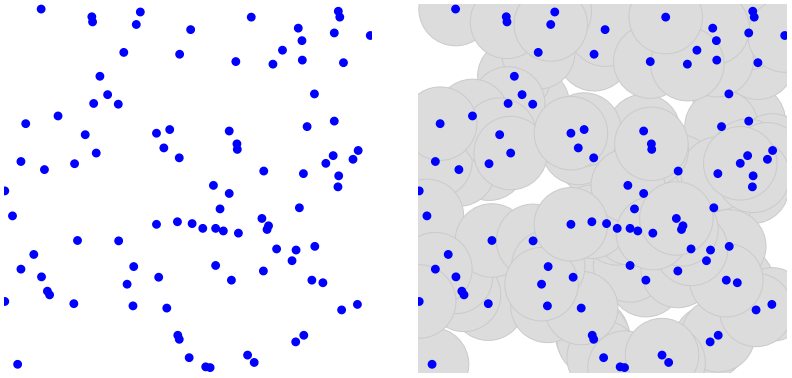
A geometric random graph will be denoted $GRG(n, r)$, where r is the radius of a circle around each node and n is as defined previously. The graph is constructed in three phases with an example shown in Figure 2.1.

1. Place n nodes randomly in a unit square
2. Draw a circle of radius r around each node
3. If two nodes fall into each other's neighbourhoods, then an edge is placed between the two nodes

Although this means that the graph is bounded, its structure is much more closely related to that of real wireless networks. This is because nodes only affect nodes which are within their neighbourhood and for a wireless device the size of the neighbourhood

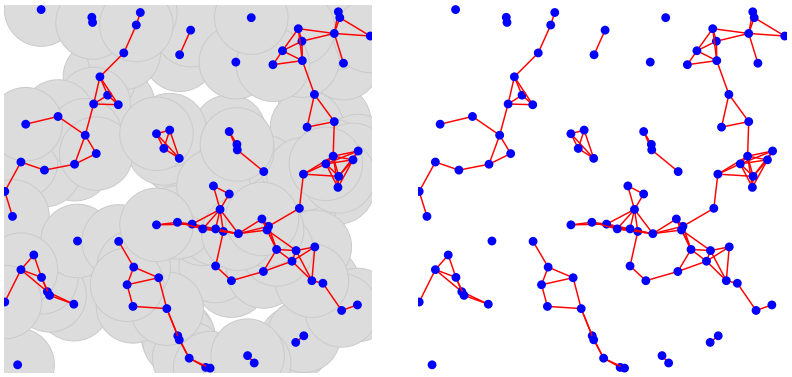
can be seen as its transmission power level, where a higher power setting implies a greater likelihood of more interference.

As well as $GRG(n,r)$, it will also be denoted by $GRG\{n,\overline{d}\}$, where $r = \sqrt{\overline{d}/(n\pi)}$.



(a) Place n=100 nodes randomly

(b) Place circles radius r=0.1



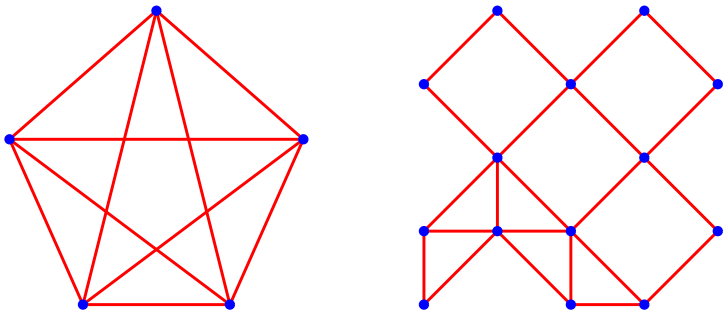
(c) Connect neighbouring nodes

(d) Final graph

Figure 2.1: Construction of a geometric random graph with 100 nodes and radius of 0.1, $GRG(100, 0.1)$.

2.6.3 Planar graphs

A planar graph has at least one embedding on the plane where no two edges cross. Due to this it is easy to show a graph is planar with only one such construction required. However, to show a graph is non-planar requires every combination of placements to be tested. The only non-planar graph of five or less nodes is K_5, the complete five-node graph, where all nodes are connected to each other. In this report a planar graph will



(a) Non-planar (b) Planar

be denoted $PL(n)$, where n is the number of nodes in the graph. Three methods for generating planar graphs are investigated in this report.

1. Generate a number of $G(n, p)$, and filter out the non-planar graphs
2. Markov chain algorithm by Denise et al [9]
3. Street graph (see section 2.6.4)

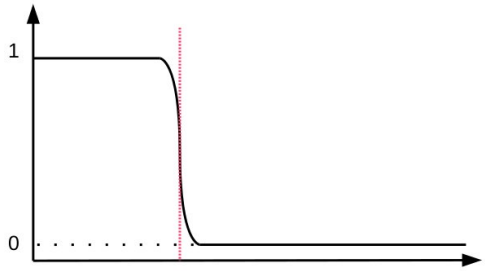


Figure 2.2: Fraction of planar graphs; horizontal axis: density; vertical axis: fraction of Erdős-Rényi graphs which are planar.

The first method highlighted has two major flaws, which are represented in Figure 2.2. At a low density the vast majority of graphs produced are planar, however, these are generally very sparse with minimal connectivity. This means they provide no useful insight when studying them. At a high density, only a minimal fraction of the graphs generated are planar and, although these would be useful to study, the process of generating them is too inefficient. There is a critical density, shown by the red line in Figure 2.2, at which the fraction of planar graphs produced is acceptable and

of interest to study. However, finding this density is a hard problem, so the second method is preferred.

The Markov chain algorithm works by randomly adding or removing edges to a graph. At each time step, two nodes are selected and their current connection is inverted, so an edge is added if there was previously no edge and vice versa. If this change preserves planarity, which requires a checker to confirm, then the change is accepted. This process then carries on working iteratively over time. To obtain different random planar graphs the current structure is taken at given time intervals.

2.6.4 Street graphs

Street graphs were designed by Keith Briggs and are constructed in such a way that they are always planar. This means that they exhibit all of the properties that planar graphs possess. One extra feature is that every node in a street graph has $\deg(i)$=2, 3, or 4 with probability one. It is constructed inside the unit square as follows and will be denoted $ST(l)$.

1. Place l points uniformly distributed in the unit square
2. Insert a line going through each point at a random uniformly distributed angle
3. Remove the l points used to position the lines
4. Where two lines intersect inside the unit square place a node
5. Place an edge along any line that connects two nodes

The final version of an $ST(20)$ is shown in Figure 2.3.

The chance of three or more lines intersecting at the same point is considered to have probability zero, so this outcome is ignored. The graph takes its name as it can be seen to replicate the road layout of a city, with edges representing roads and nodes intersections. This is based on the assumption that roads are infinite in length and straight. To achieve the greatest coverage along the lines, wireless access points would be placed where the nodes occur.

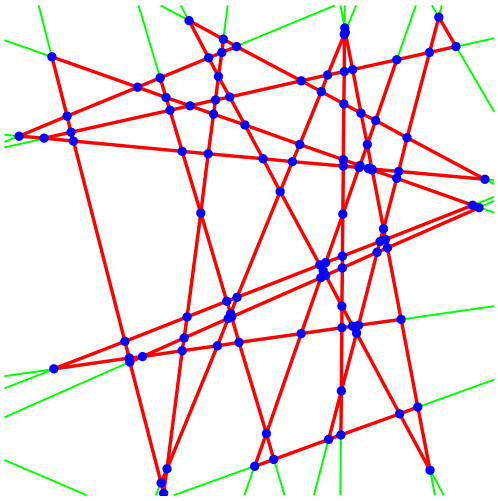


Figure 2.3: Street graph with 20 lines; green lines are the original lines used to construct the graph; red lines indicate edges in the graph; blue dots represent nodes.

2.6.5 Graph colouring

Proper colouring

A proper colouring for a graph is achieved when no edge connects two nodes which have the same colour. This means that there are no conflicts in the graph, so no interference in a wireless network and this is the ideal solution to achieve. However, this may not always be possible, particularly with only local knowledge.

Chromatic number

The chromatic number of a graph is the smallest number of colours for which a proper colouring can be achieved and is denoted by χ. To find this is an NP-hard problem which requires global knowledge of a graph to solve.

For any graph it has been proved that $\chi \leqslant \Delta+1$, where Δ is the highest degree of a node in the graph, although this is generally considered a relatively poor upper bound. For large $G(n,p)$ graphs, Achlioptas and Naor [6] showed that χ can be accurately predicted as either k or $k+1$ where k is the smallest integer such that $d < 2k\log(k)$, for a $G(n,d/n)$ graph. It has been shown that this means k is given by $\lceil d/(2\mathrm{W}(d/2))\rceil$ [2], where W denotes the Lambert W function, which is defined as the inverse of $x \to x\exp(x)$. For large sparse geometric random graphs it has been shown that χ and Δ converge to the same number [14], although global knowledge is still required to compute this.

For planar graphs, and therefore street graphs, $\chi \leqslant 4$. Due to this, these graphs will be used to test how effective the heuristics are at finding proper colourings when C is close to χ.

Graphs can be categorized in the following three ways:

- Under-constrained: $C > \chi$
- Critically-constrained: $C = \chi$
- Over-constrained: $C < \chi$

For under-constrained graphs, it is expected that the heuristics would be able to return proper colourings most of the time, therefore fully optimizing the network. For critically and over-constrained graphs, acceptable resolutions rather than a proper colouring would be expected.

Chapter 3

Distributed asynchronous heuristics

One of the main challenges in programming the heuristics for this report is using one machine to mimic the behaviour of many individual processors. To achieve this an event queue is used, where each event contains two pieces of information which allow for the distributed asynchronous nature to be achieved. These are

- τ - Next time the given node is due to activate
- i - Node to activate

where τ is used as the key field to sort the event queue. Each event is then effectively telling the machine which device it is acting as at that given point in time.

3.1 Asynchronous time

To achieve asynchronous time the Poisson process is used where events occur independently of each other at rate λ per unit time. This is done using the exponential distribution with probability density function

$$f_\lambda(\tau) = \lambda \exp(-\lambda\tau). \tag{3.1}$$

τ is then determined by inverting the cumulative pdf, meaning

$$\tau = -\frac{1}{\lambda}\ln(1-x) \tag{3.2}$$

where $x \sim U[0,1)$, so x is uniformly distributed between 0 and 1, and λ is the desired rate, with $\lambda > 0$. This generating function is equivalent to the expovariate function

from the python library [3].

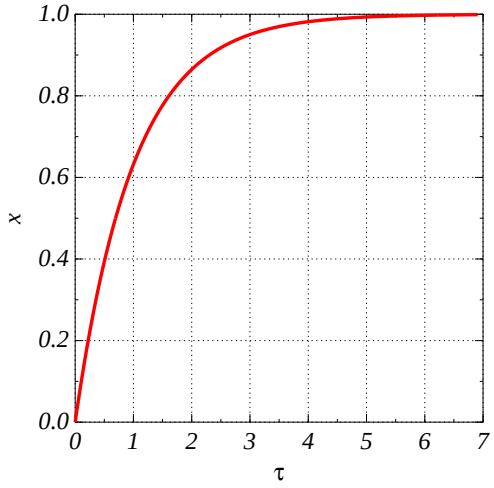


Figure 3.1: Cumulative pdf of exponential distribution with λ=1.

3.2 Initialization and optimization

Algorithm 1 Initialization of node i

Initialization:
const colours $\leftarrow \{0,\ldots,C-1\}$
$c_i \leftarrow 0$
$\tau \leftarrow$ expovariate(1)

Each graph is initialized with n nodes and stored using a graph library created by Keith Briggs [4]. Once created the structure of the graph is then formed depending upon which classification of graph is desired. This involves inserting the edges of the graph. All devices in the network are then initialized in the same way and this is shown in Algorithm 1. They are programmed with the number of channels available for use, which would usually be done during the manufacturing process of the device. When the device is switched on for the first time it sets itself to channel 0 and assigns itself an activation time τ using the expovariate generating function in equation 3.2 with λ=1. For all simulations, devices are first switched on at time t=0, meaning every device will be on channel 0, so initially every device is in conflict with its neighbours, so the total number of conflicts in the system is equal to the number of edges. Finally the event queue is then initialized for all $i \in$ Ngraph(0) with their respective τ values.

The network is then optimized over a length of time, given either by a number of checks or run time, for a set number of channels. For each check the top event is then removed from the queue. The node with which this corresponds becomes active and carries out the check function. A new τ is then assigned to that node from a new randomly produced time from equation 3.2 that is added to the current time of the simulation. This event is then inserted back into the queue at the correct position.

3.3 Check function

For this report it is the autonomous behaviour of each device that will be considered, with different heuristics used in an attempt to obtain improved results and performance. The heuristics will be classified using the following notation.

- rate — fixed/variable — F/V
- resolution — complete/partial/mixed — C/P/M
- choice — deterministic/stochastic — D/S

Here *rate* refers to how regularly a device will check itself for conflicts to try and improve its current state. Fixed implies each device is equally likely to check itself, while variable rates will introduce the idea of 'targeting', with devices with more conflicts more likely to check themselves.

Resolution looks at how a device reduces its conflicts. Complete means a device is only willing to move to a channel free of interference. In partial resolution heuristics devices move to the channel with the lowest level of interference. Mixed provides an intermediary where devices will try and improve their state, though if this is not possible, are willing to move in an 'uphill' direction to try and help the overall performance of the network.

Choice relates to how a device chooses its new channel. Deterministic means a device assigns itself to the first channel it finds at a given interference level. For stochastic choice a device randomly assigns itself a channel, either from the whole range or a subset which meets specified conditions, allowing a network to stay 'fresh' and prevents it becoming stuck in a sub-optimal state.

All of the heuristics will use the *number_of_conflicts* function which calculates the level of interference a specified channel has. It compares a channel to the usage of all channels a device observes in its local environment at that given point in time. The

Algorithm 2 number_of_conflicts(col, col_neigh)

```
conflicts ← 0
for j ∈ col_neigh do
    if col = j then
        conflicts ← conflicts+1

return conflicts
```

code is shown in Algorithm 2. This device compiles this by listening into all of the channels to determine their current usage in the device's local neighbourhood. With devices now having the ability to do this, it means no messages have to be passed between devices to enable channel allocation. This means no message queues are built up caused by a device having to respond to many requests for their current channel. This technology enables pure locality as well, with devices not having to know where their neighbours are and on which channel each is specifically operating. An example is shown in Figure 3.2.

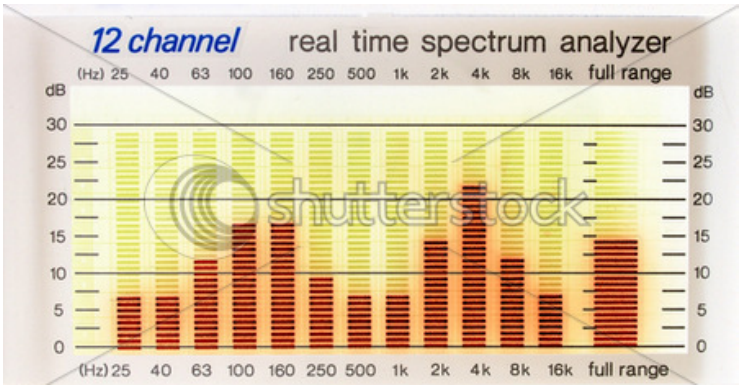


Figure 3.2: Channel usage [5].

3.4 Performance measures

All of the performance measures are collated using global knowledge, however each device had no access to this knowledge during simulations.

Checks — ρ

The number of checks is used when looking at the efficiency of a heuristic, with lower values signaling better results. It will be denoted by ρ and is normalized so that graphs of varying size can be compared. This means for a graph with 100 nodes, after 100 checks ρ=1. The theory behind this is that if a device is checking for conflicts then it is spending time not performing its primary role. Therefore, by reducing these checks while still optimizing the network efficiently, it enables costs to be reduced. The check function also drains power from battery operated devices, so reducing checks implies longer battery life.

Completion — ψ

When looking at under or critically-constrained graphs, the main measure to be used will be how successful the heuristics are at finding a proper colouring, which will be denoted by ψ. It is defined as the fraction of runs for which a proper colouring is found, so takes values in the range $[0, 1]$.

Conflicts in network — κ

Conflicts, denoted κ, will be used for over-constrained graphs, or when χ is unknown. It is initially defined as the sum of edges $i \leftrightarrow j$ st c_i=c_j, but will be normalized in two ways. The first is by dividing by the number of edges a graph has. This is so graphs of different sizes can be compared. The second is by multiplying by the number of channels available for use, C. This means it takes values of $[0, C]$. This is because heuristics with larger C values would be expected to demonstrate better performance than small C values on the same graph. This result is the performance measure used by Fitzpatrick and Meertens [12].

$$\kappa = C \frac{\sum_{\text{Egraph}(t)} c_i = c_j}{\sum_{\text{Egraph}(t)} i \leftrightarrow j} \tag{3.3}$$

κ will be measured in different ways and these will be shown using a subscript notation. The first is at a given point in time, usually occurring at the end of a simulation and this will be denoted κ_t. For simulations where κ tends towards a stabilized value, then this can be seen as κ_∞, being the number of relative conflicts in the network if it ran indefinitely. It can also be averaged over the length of a simulation, with a new

value occurring every time a device checks itself. Here it will be denoted $\kappa_{t_0-t_1}$, where t_0 is the clock time at which the averaging starts and t_1 where it ends.

Conflicts per device — δ

This performance measure calculates the average level of interference per device and will be denoted δ. It will be especially useful for judging networks as they change over time. This is defined as the sum of each devices' conflicts, $\text{conf}(i)$, divided by the total number of devices currently switched on, with the formula shown in equation 3.4.

$$\delta = \frac{\sum_{i \in \text{Ngraph}(t)} \text{conf}(i)}{\text{size}(\text{Ngraph}(t))} \tag{3.4}$$

3.5 Rejected heuristics

One set of heuristics considered was conflict creation. Here a device would assign itself two activation points in time, one for conflict resolution and one for conflict creation where it would deliberately change channel so that it had more conflicts. This was to try and introduce the 'uphill' idea but the stochastic choice principle, when no better resolution is available, performed better.

When looking at variable rates, a sample which were related to the degree of a device were considered. This was to help the 'targeting' effect but these proved to be very inefficient. This is because a device with many neighbours but no conflicts would remain on a very high check rate. It also requires a device to know how many neighbours it has, and this would require message passing which is to be avoided.

Chapter 4

Rate heuristics

The first and simplest heuristic to be studied in this report is FCD, with the structure of the heuristic shown in Heuristic 3. When a device reaches its next given check time, τ, it becomes active and carries out a given set of instructions. It starts off by listening into all of the channels in its local neighbourhood to form the set *col_neigh*. If no neighbours are found then the device reassigns itself to channel 0 and computes a new τ. If other transmissions are sensed then the device cycles through all available channels, assigning itself to the first free channel it finds. If a device cannot detect any free channels then it will remain on the channel it was using when the checking process started.

Heuristic 3 FCD heuristic for node i; fixed rate, complete resolution only, deterministic choice; fixed check rate of λ=1

```
Action:
  if clock_time = τ then
      col_neigh ← {colour of neighbours}
      if col_neigh = ∅ then
          c_i ← 0
      else
          for col ∈ colours do
              if number_of_conflicts(col, col_neigh) = 0 then
                  c_i ← col
                  break for
      τ ← τ+expovariate(1)
```

The FCD heuristic can be viewed as greedy, with devices operating self-centredly, as they are only willing to change channel if it provides maximal improvement to

their own performance. This means a network can potentially become stuck in a sub-optimal state. This is where the 'uphill' idea from simulated annealing will be looked at and adapted. There is also no sense of 'targeting' in the FCD heuristic, with every node in the network equally likely to check for interference. In reality devices at the centre of the network should be able to check more often as they are likely to have more neighbours and thus a greater chance of been involved in a conflict. Therefore by allowing these devices to check more often, optimal results for the network should be achieved more efficiently and this heuristic will be looked at later in this chapter.

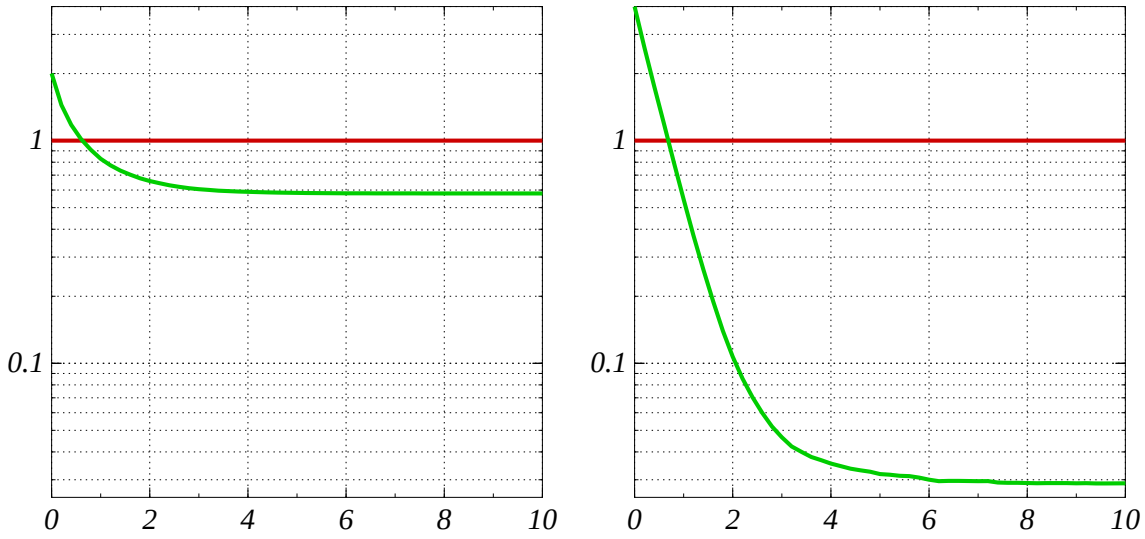


Figure 4.1: Performance of FCD heuristic with check rate of λ=1 and a random colourer on an Erdős-Rényi graph with 100 nodes and mean degree of 3; horizontal axis: run time; vertical axis: relative conflicts (κ_t); 2000 simulations with a run time of 20; left: channels C=$\chi-1$; right: C=$\chi+1$; red: random colourer; green: FCD with λ=1.

For all the results considered in Chapters 4 to 6 a constant graph process is used, so $\text{graph}(0) = \text{graph}(1) = \text{graph}(t)$. This is equivalent to devices remaining in a fixed position and constantly transmitting.

The first result considered compares the FCD heuristic against a random colourer with the aim of showing that even the simplest of local distributed asynchronous heuristics prove beneficial to resolving interference in a network. The random colourer works by each device assigning itself a channel between 0 and $C-1$ using a uniform probability distribution. Devices then remain on this channel for the entire run time, so no optimization of the network occurs and the network has an expected value of κ_∞=1. Therefore the FCD heuristic is expected to outperform the random colourer.

For this test the Erdős-Rényi model is used. Graphs with the same structure were looked at with a different number of channels available for a device to use in each case and the results are shown in Figure 4.1. The graphs were created with 100 nodes and a mean degree of 3, $G\{100,3\}$. In the left plot a device has $\chi-1$ channels to use and $\chi+1$ on the right. 2000 simulations were used and averaged with a run time of 20 in each case.

Figure 4.1 clearly shows that as time progresses the FCD heuristics produce better results for the network in terms of reducing interference. However both graphs leveled off in conflicts, κ, after run time exceeds 10, so this has been excluded from the plots. The simulations with $\chi+1$ channels available performed the best despite the performance measure being normalized. This is because more channels provides greater scope for optimization away from the random colourer. However, given this network is under-constrained complete elimination of κ on every run is desired, but 15% of the time this did not happen as networks became stuck in sub-optimal states. Choice heuristics in Chapter 5 look at improving this. It can also be seen that when devices were allowed more channels, the network required a longer time frame to reach a resolution. This is due to the greater scope for optimization as mentioned before, so the optimization process occurs for longer.

The initial poor performance of the FCD is due to the way devices are initialized, all with c_i=0, meaning the entire network starts in conflict. In real life all devices would not be initialized at the same point in time, with devices being gradually introduced to the network, so the first part of the graph is of little interest. Focus is on the performance of the network as time progresses.

The next result compares Erdős-Rényi graphs, $G\{n,\overline{d}\}$, and geometric random graphs, $GRG\{n,\overline{d}\}$. These were both tested with n taking values of 50 and 100. This was to allow χ to be computed efficiently [4]. The graphs were then given a mean degree ranging from 3 to 10. For each combination of n and $\overline{d}$, 1000 simulations were run and the results averaged. Each simulation had a run time of 20. The simulations were run with a varying number of channels. For the left plots in Figures 4.3 to 4.5 C=$\chi-1$, χ or $\chi+1$, and the right plots show fixed C, ranging from 2 to 8. Figure 4.3 displays how κ_{20} performs against $\overline{d}$, with results from a run only included in the average when a proper colouring is not found in the given time frame. Figure 4.4 shows the fraction of runs for which a proper colouring was found in the time allowed. Figure 4.5 provides a guide to the combined performance of the heuristics with κ being weighted by the inclusion of κ=0 in the averaging when a proper colouring is found. For the $G\{n,\overline{d}\}$ simulations n=1000 was also tried, with χ computed using the approximation

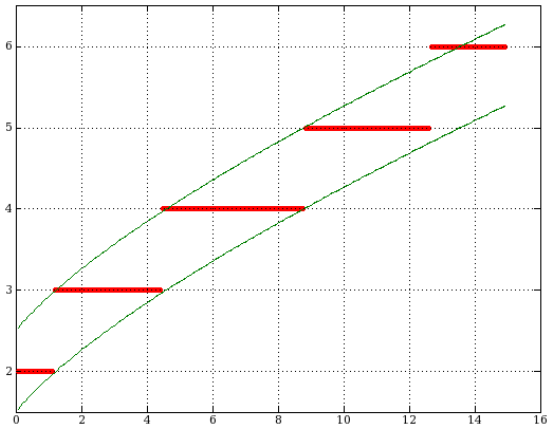


Figure 4.2: Chromatic number graph for $G(200, d/200)$, where $d \approx$ mean degree; horizontal axis: d; vertical axis: predicted chromatic number [2].

mentioned in section 2.6.5.

On the plots shown in Figure 4.3 it is seen that for almost all the values of C chosen that the FCD heuristics outperform the random colourer, which would be seen at 1 on the plots. For the fixed channel plots on the right, as $\overline{d}$ increases the heuristics approach κ_{20}=1 and go above this in some cases. This is because for this heuristic the network can easily get stuck in a sub-optimal state and this happens more frequently as networks become more populated. For the results considered here the networks become stuck in this sub-optimal state by time 20, so κ_{20} can be seen as κ_∞.

For the three values of C chosen with varying χ (left plots in Figure 4.3), very different results between the $G(n,p)$ and $GRG(n,r)$ graphs are observed. For $G(n,p)$ there are peaks in the curves, with these becoming more pronounced as n increases. This is because the χ value for sparse $G(n,p)$ graphs increases in clear steps, with this feature becoming more prominent as $n \to \infty$. This result is shown in Figure 4.2, with the step up in chromatic number occurring at the same mean degree as the peaks displayed in the FCD heuristics results. For the $GRG(n,r)$ graphs the level of interference remains constant as the mean degree increases for fixed n.

When comparing the plots with fixed C, both $G(n,p)$ and $GRG(n,r)$ now display the same shape, with the best results for each C been achieved at the lower values of m. For the same value of C it can be seen that $G(n,p)$ achieves fewer conflicts than $GRG(n,r)$. This is due to the more complex structure of the $GRG(n,r)$ graph which means it has a higher value of χ for the same mean degree as $G(n,p)$. For the lower values of m the conflicts tend to a given value of κ for varying n. This is because the graphs are over-constrained, so these values are equivalent to one conflict in the

network, which is the best result that can be achieved without a proper colouring being found.

It can also be seen that for over-constrained graphs the values of κ tend towards each other as n increases for a fixed number of channels, so n can be ignored when trying to predict κ. This is clearly seen with the lines displaying varying n lying on top of each other.

When studying the completion plots shown in Figure 4.4 there is again a big difference between the $G(n,p)$ and $GRG(n,r)$ graphs. The $G(n,p)$ plots again display the peak and troughs pattern which was seen in the conflict plots. The peaks come when the value of χ has just increased. This is because a greater number of different proper colourings exist for a graph when χ has just increased, making it easier to find a proper colouring. For $GRG(n,r)$ there is a decline in ψ as $\overline{d}$ increases, with this decline occurring more rapidly when devices had a fixed number of channels to use. This shows it is harder for the heuristics to find a proper colouring as n increases.

Comparison between the two graph types shows that for the plots where C alters with χ that the best results are achieved on the $GRG(n,r)$. This is because the $GRG(n,r)$ graph can be split into smaller sub-graphs which aids solving and this is due to the small numbers of nodes been used, where in $G(n,p)$ the graph is more likely to be connected. It is again seen that for the same fixed value of C, results were better for the $G(n,p)$ graphs and this is due to the reasons discussed before.

For both graphs it is clear to see that the FCD heuristic achieves a proper colouring on a greater fraction of runs for graphs of smaller size. This means that n cannot be discarded when trying to predict the value of ψ.

The plots in Figure 4.5 clearly display the overall performance of the heuristics for the given the number of channels available. The drop-off of κ_{20} at the lower mean degrees shows that for these networks an optimal resolution was found for a larger volume of runs with the elimination of all interference in the networks. For a given value of C, n can be ignored as $n \to \infty$ with all the lines tending towards each other.

To try and achieve the notion of the 'targeting' effect, heuristics using variable rates, labeled VCD, will be tested. The aim with these is to improve the efficiency of the heuristic, meaning the network is resolved to an acceptable standard using the smallest number of checks. This means the network spends less time in a sub-optimal state and devices are able to spend more time carrying out their primary role rather than resolving conflicts.

The variable rate is introduced when a device assigns itself a new value of λ and thus τ. An example of one VCD heuristic where $\lambda = 1 + \text{conf}(i)$ is shown in Heuristic 4.

Heuristic 4 VCD heuristic for node i; variable rate, complete resolution only, deterministic choice; check rate λ=1+conf(i)

Action:
  if clock_time $= \tau$ **then**
    col_neigh $\leftarrow$ {colour of neighbours}
    if col_neigh $= \emptyset$ **then**
      $c_i \leftarrow 0$
      conflicts $\leftarrow 0$
    else
      for col $\in$ colours$\setminus\{c_i\}$ **do**
        conflicts $\leftarrow$ number_of_conflicts(col, col_neigh)
        if conflicts $= 0$ **then**
          $c_i \leftarrow$ col
          break for
    $\tau \leftarrow \tau$+expovariate(1+conflicts)

The other rates that will be tested are:

- $\lambda = \text{conf}(i)$
- $\lambda = \exp(\text{conf}(i))$

and these will all be compared against the FCD heuristic with λ=1. λ=1+conf(i) was chosen as it provides a linear growth of λ as the number of conflicts a device has grows. The exponential rate is used to see whether optimizing the 'targeting' effect helps the performance of the heuristics. The λ=conf(i) rate again provides linear growth, but requires special attention when a device has no conflicts as it will assign λ a value of 0, which equation 3.2 does not allow. To overcome this, when a device resolves all of its conflicts it will not set a new check time, effectively removing itself from the event queue. This means it will rely on its neighbours to resolve any conflicts which it may gain.

Throughout the rest of this report these variable rates will be trialled to see whether the notion of 'targeting' is beneficial in networks.

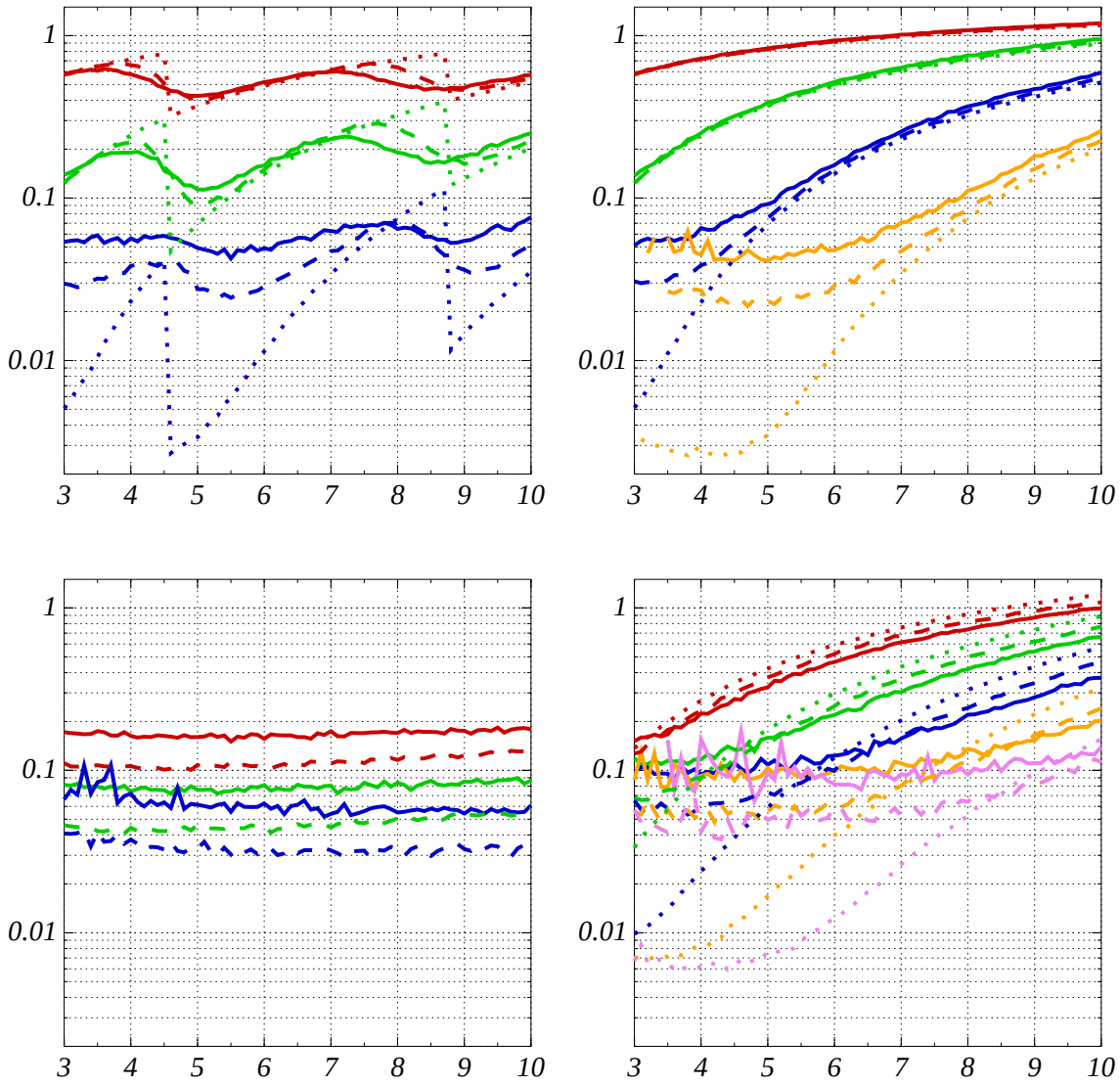


Figure 4.3: Relative conflicts at time 20 compared to mean degree with graphs using a different number of nodes (n); horizontal axis: mean degree ($\overline{d}$); vertical axis: relative conflicts (κ_{20}) with a $\log_{10}$ scale excluding κ=0; 1000 simulations with a run time of 20; top: Erdős-Rényi model, $G\{n,\overline{d}\}$; bottom: geometric random graph, $GRG\{n,\overline{d}\}$; left: red: channels C=$\chi-1$, green: C=χ, blue: C=$\chi+1$; top right: red: C=2, green: C=3, blue: C=4, orange: C=5; bottom right: red: C=4, green: C=5, blue: C=6, orange: C=7, violet: C=8; solid: n=50, dashed: n=100, dotted: n=1000 (excluded from bottom left plot).

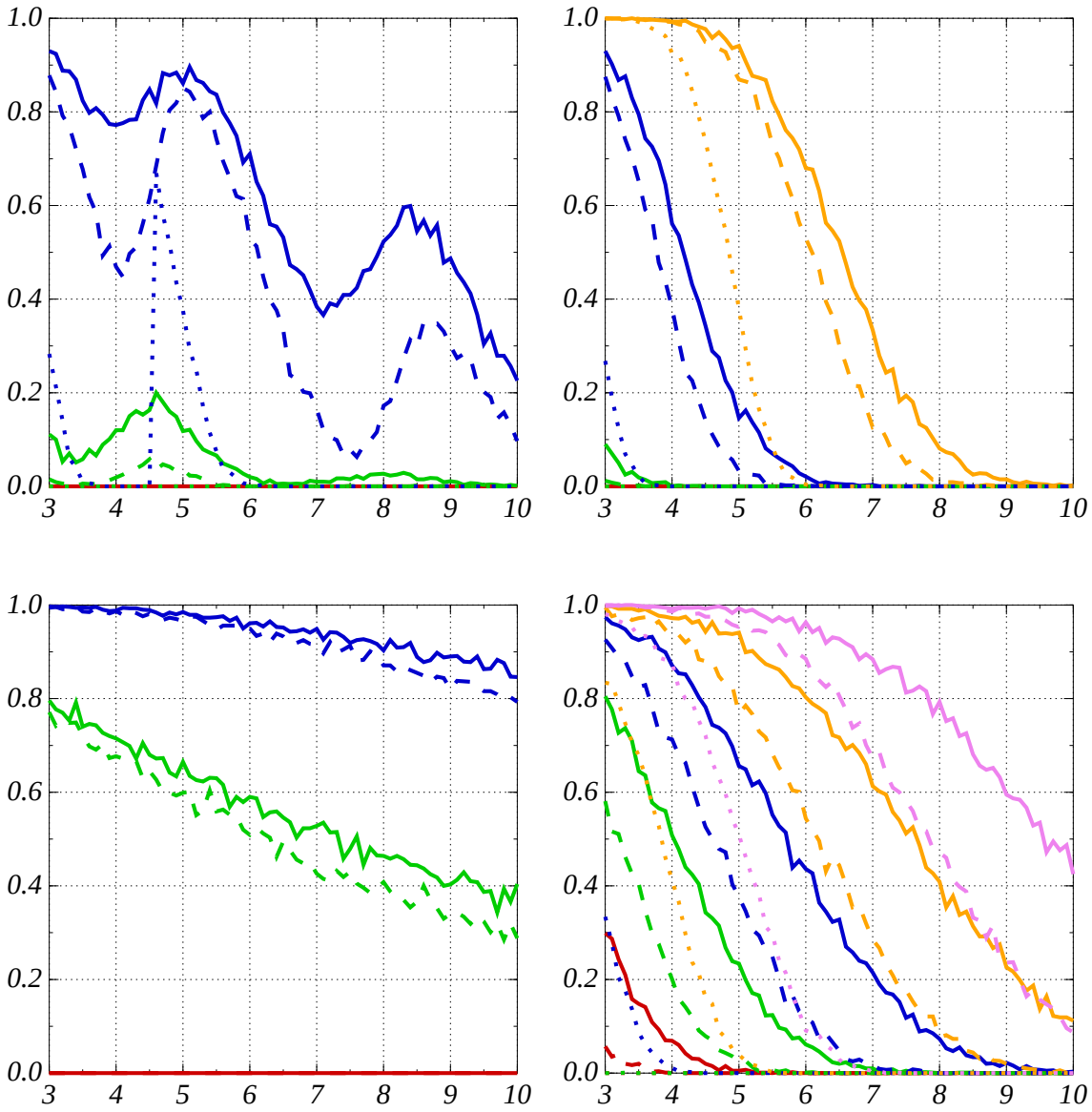


Figure 4.4: Fraction of runs for which a proper colouring was found compared to mean degree with graphs using a different number of nodes (n); horizontal axis: mean degree ($\bar{d}$); vertical axis: fraction of runs for which a proper colouring was found (ψ); 1000 simulations with a run time of 20; top: Erdős-Rényi model, $G\{n,\bar{d}\}$; bottom: geometric random graph, $GRG\{n,\bar{d}\}$; left: red: channels C=$\chi-1$, green: C=χ, blue: C=$\chi+1$; top right: red: C=2, green: C=3, blue: C=4, orange: C=5; bottom right: red: C=4, green: C=5, blue: C=6, orange: C=7, violet: C=8; solid: n=50, dashed: n=100, dotted: n=1000 (excluded from bottom left plot).

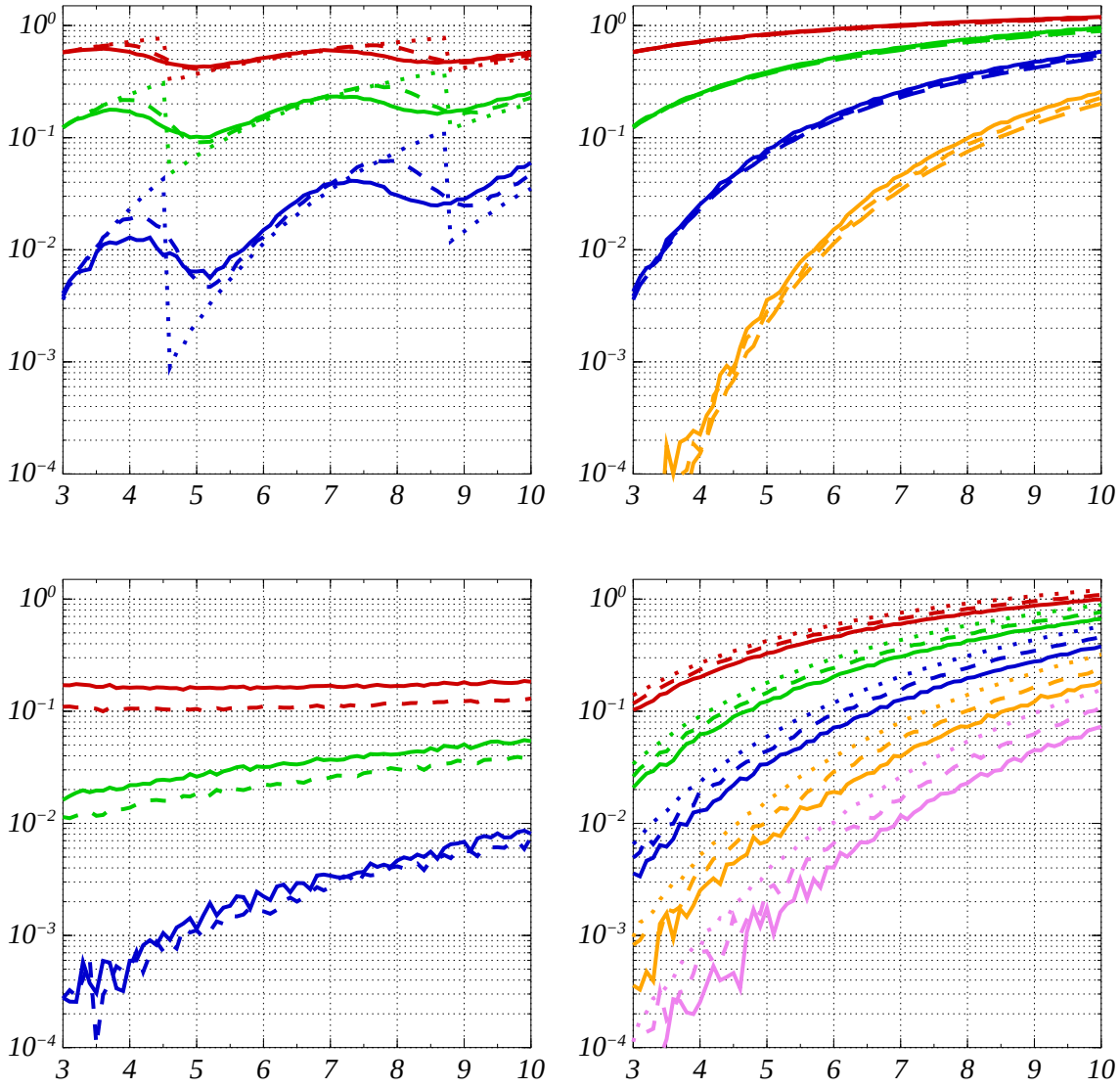


Figure 4.5: Relative conflicts at time 20 compared to mean degree with graphs using a different number of nodes (n); horizontal axis: mean degree ($\overline{d}$); vertical axis: relative conflicts (κ_{20}) with a $\log_{10}$ scale including κ=0; 1000 simulations with a run time of 20; top: Erdős-Rényi model, $G\{n,\overline{d}\}$; bottom: geometric random graph, $GRG\{n,\overline{d}\}$; left: red: channels C=$\chi-1$, green: C=χ, blue: C=$\chi+1$; top right: red: C=2, green: C=3 blue: C=4, orange: C=5; bottom right: red: C=4, green: C=5, blue: C=6, orange: C=7, violet: C=8; solid: n=50, dashed: n=100, dotted: n=1000 (excluded from bottom left plot).

Chapter 5

Choice heuristics

This section introduces different ways in which a device assigns itself a new channel and this is done using VCS heuristics. The idea behind this is to keep a network 'fresh' by changing the conflicts a device has, preventing the network becoming stuck in a sub-optimal solution. An example is shown in Figure 5.1 where C=3.

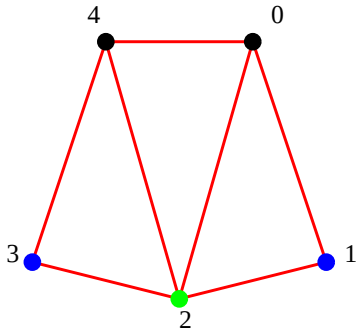


Figure 5.1: Sub-optimal graph resolution; channels 0, 1 and 2 are represented by the colours black, green and blue respectively.

When nodes 0 and 4 check for conflicts they both find no free channels and thus remain on channel 0, meaning $0 \leftrightarrow 4$ remains in conflict. Introducing a stochastic choice to channel assignment means node 0 now has probability of $1/3$ of changing to channel 2. Now, when node 1 checks its conflicts, it finds channel 2 is now taken but channel 0 is free and thus a proper colouring of the graph is achieved.

For the VCS heuristic, shown in Heuristic 5, the stochastic choice is only introduced when there are no empty channels available. Now, instead of remaining on the same channel a device will randomly choose a new channel on which to operate, with this

being decided using $U[0, C-1]$, the uniform distribution of integers between 0 and C-1. This now means devices are behaving in a more altruistic fashion as they are willing to change channel even if they receive no benefit in the short run. In some cases a device can end up with more conflicts than before, which is the idea used in simulated annealing.

Heuristic 5 VCS heuristic for node i; variable rate, complete resolution only, stochastic choice; check rate λ=1+conf(i)

```
Action:
  if clock_time = τ then
      col_neigh ← {colour of neighbours}
      if col_neigh = ∅ then
          c_i ← 0
      else
          for col ∈ colours do
              conflicts ← number_of_conflicts(col, col_neigh)
              if conflicts = 0 then
                  c_i ← col
                  break for
          if conflicts > 0 then
              c_i ← uniform_discrete[0, C−1]
      τ ← τ+expovariate(1+number_of_conflicts(c_i, col_neigh))
```

To test the VCS heuristics against the VCD heuristics street graphs will be used. This is because $\chi \leqslant 4$, so devices can be initialized with knowledge that a proper colouring exists. This means the heuristics will be judged using ψ, the fraction of runs for which a proper colouring is achieved. The street graphs used are created using 40 lines, denoted $ST(40)$. This equates to graphs having approximately 440 nodes with a mean degree of 3.8. 1000 simulations were tested with a run time of 100. The results are shown in Figure 5.2, with the left plots displaying the VCD heuristics and the VCS heuristics on the right. The top plots use ψ as the performance measure, while the bottom plots use a $\log_{10}(1-\psi)$ scale. This enables us to determine the rate at which the proper colourings are achieved. Extra data for these simulations is shown in Table 5.1.

When looking at the street graph results and comparing the VCD and VCS heuristics, the introduction of the stochastic choice to channel assignment produces a 100% success rate at finding a proper colouring when a device still checks itself with no conflicts. This is because networks were able to go in an 'uphill' direction meaning

they did not become stuck in sub-optimal states, as happened with the VCD heuristics. Here it is seen that by ρ=15 the networks have become stuck in a sub-optimal state and remain there indefinitely, so only this area is shown in the plots.

The $\log_{10}(1-x)$ scale plots in Figure 5.2 enables us to gain information about the rate at which a proper colouring is found for each simulation. For the stochastic choice heuristics the plots display straight lines which signify that optimal solutions were obtained at an exponential rate. For the deterministic choice heuristics the rate at which a proper colouring is found was slower, demonstrated by the shallower gradient of the lines, with these quickly flattening out showing that all unresolved networks became stabilized in a sub-optimal state.

		ψ	ρ	time	ρ_∞	κ_{20-100}
λ=1	FCD	0.553	4.886	4.892	100.012	0.00647
	FCS	1	4.980	4.979	-	-
λ=1+conf(i)	VCD	0.531	4.900	4.863	100.630	0.00631
	VCS	1	4.787	4.748	-	-
λ=conf(i)	VCD	0.217	1.025	4.606	1.887	0.00809
	VCS	0.005	1.016	4.730	2.327	0.02762
λ=exp(conf(i))	VCD	0.582	4.873	4.816	101.061	0.00614
	VCS	1	4.750	4.697	-	-

Table 5.1: Data from simulations on street graphs with 40 lines, $SL(40)$; 1000 simulations with a run time of 100 for each heuristic; ψ shows the fraction of runs for which a proper colouring was found; ρ displays the average number of relative checks used when a proper colouring was found; t represents the average run time when a proper colouring was found; ρ_∞ is the number of checks when a proper colouring was not achieved; κ_{20-100} is the average level of interference in the network between t=20 and 100.

When comparing the varying check rates used, two distinct performances are seen when looking at the fraction of runs for which a proper colouring is achieved. The better performance results are achieved when a device with no conflicts still checks itself. The three rates, λ=1, 1+conf(i), exp(conf(i)), have virtually identical performance across all performance measures.

The λ=conf(i) check rate demonstrates a completely different performance across all measures and leads to considering a trade-off between ψ and ρ. Here it is seen that the fraction of runs for which a proper colouring was found is very poor and also that

networks have a higher level of interference, although this is still at an almost negligible level. This is because the majority of devices are resolving their conflicts and then not checking again for the rest of the simulation. This means only a few devices are left trying to resolve the network and this leads to sub-optimal performance in κ and ψ. However, the number of relative checks is significantly lower and for mobile devices running on battery power this is very important. This is because when a device checks for conflicts it uses up battery life, so reducing these checks results in longer battery life and a happier user.

Another noticeable difference with the λ=conf(i) check rate is that the VCD heuristic performs better than the VCS heuristic in terms of ψ. This is because the stochastic choice means that devices which had no conflicts, so stopped checking, can now gain conflicts but never attempt to resolve them. This means that devices can become isolated, where all the channels in their local environment are in use and all of their neighbours have at some point in time had no interference, so stopped checking. This results in some level of interference always occurring, so a proper colouring can never be achieved.

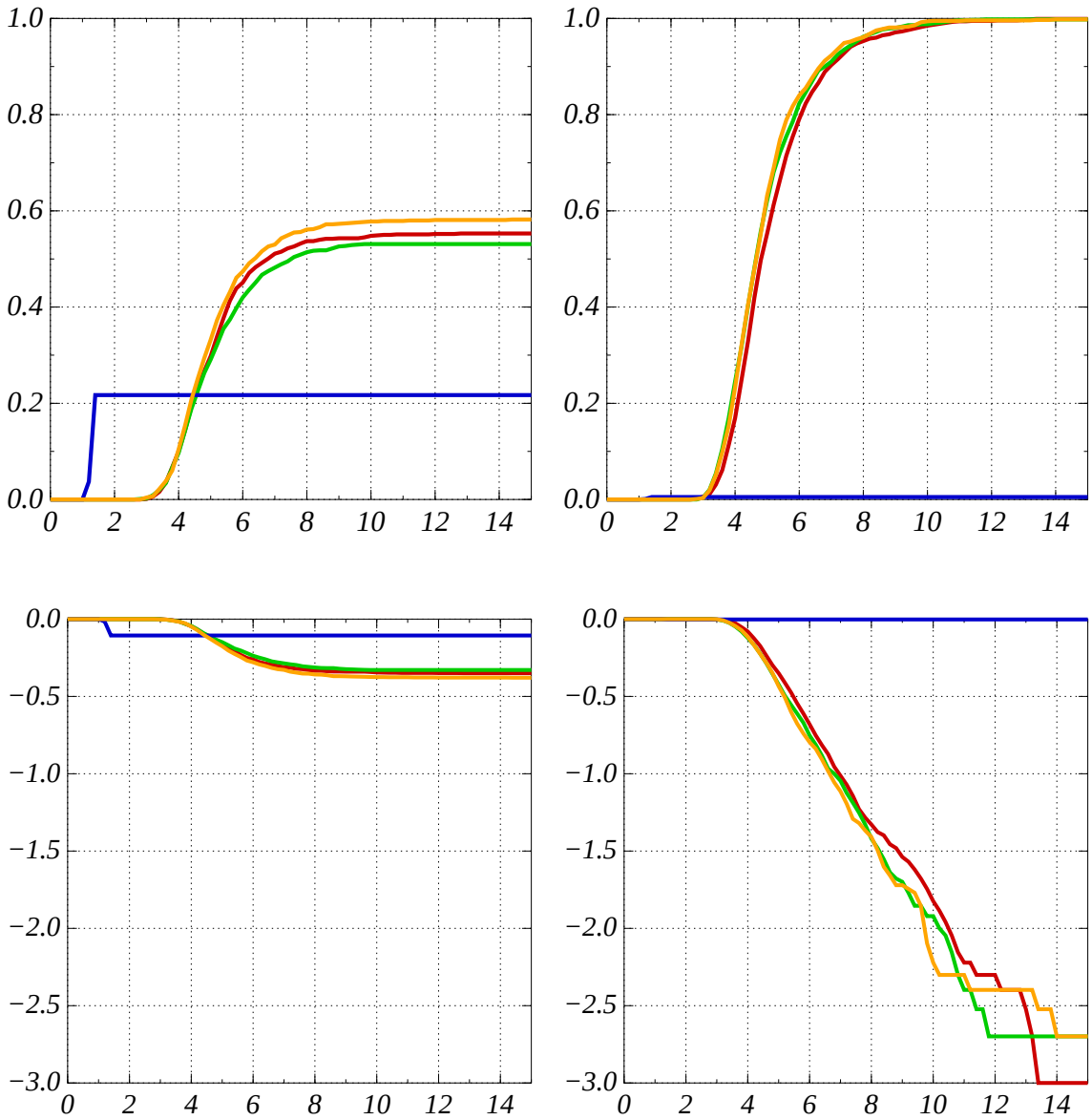


Figure 5.2: Fraction of runs for which a proper colouring was found for street graphs with 40 lines, $ST(40)$; 1000 simulations; run time of 100; devices given 4 channels; horizontal axis: relative checks (ρ); top vertical axis: fraction of runs for which a proper colouring was found (ψ); bottom vertical axis: $\log_{10}(1-\psi)$; left: FCD and VCD; right: FCS and VCS; red: check rate (λ)=1; green: λ=1+conf(i); blue: λ=conf(i); orange: λ=exp(conf(i)).

Chapter 6

Resolution heuristics

The final change to the heuristics investigates how a device resolves its conflicts. The first way this is done is partial resolution only. This can be done using both deterministic and stochastic choice. With both of these a device now chooses to operate on the channel which has the lowest level of interference. The code for VPS is displayed in Heuristic 6, and VPD operates in a similar fashion.

In VPS the device keeps track of which channels currently have the least interference and then randomly chooses a channel from this list. This is again to allow the network to stay 'fresh', but now the overall level of interference in a network cannot increase due to a check being carried out. In VPD the device chooses the channel which it first found at the lowest interference level. If no channel with strictly less interference than a device's current level is found then the device remains on its initial channel. For both cases if a free channel is found the device switches to this and ends the current checking procedure.

The second new resolution method introduced is mixed and the code is shown in Heuristic 7. It is designed to introduce an intermediate heuristic between VCS and VPS. The heuristic works by initially calculating how much interference it has on its current channel. It then cycles through all the channels and switches to a new channel if this has strictly less interference than its current channel. If a free channel is discovered then it stops the check function. If no strictly better channel is available then it uniformly chooses a new channel from all the channels it has access to, providing its current channel is not free of conflicts, in which case the devices remains as it is. This means that the heuristic predominantly works in a 'downhill' fashion, making improvements whenever it can, but also has the capability of looking 'uphill' to try and resolve the network to a more optimal standard.

Heuristic 6 VPS heuristic for node i; variable rate, partial resolution only, stochastic choice; check rate λ=1+conf(i)

Action:

```
if clock_time = τ then
    col_neigh ← {colour of neighbours}
    if col_neigh = ∅ then
        c_i ← 0
    else
        cur_col ← {c_i}
        cur_conf ← number_of_conflicts(c_i, col_neigh)
        for col ∈ colours\{c_i} do
            conflicts ← number_of_conflicts(col, col_neigh)
            if conflicts = 0 then
                c_i ← col
                break for
            else if conflicts < cur_conf then
                cur_col ← {col}
                cur_conf ← conflicts
            else if cur_conf = conflicts then
                cur_col ← {cur_col, col}
        if cur_conf > 0 then
            c_i ← uniform_random{cur_col}
    τ ← τ+expovariate(1+number_of_conflicts(c_i, col_neigh))
```

To test these new heuristics, planar graphs found using the Markov chain algorithm with 200 nodes, $PL(200)$, are used and devices are given 4 channels. This means that a proper colouring can be found, so ψ will be used as the main performance measure. On average this meant that each device had a mean degree of 4.4. 1000 simulations were used with a run time of 200. The results are displayed in Figures 6.1 and 6.2, with the later using a $\log_{10}(1-x)$ scale. The data collected for all the simulations is shown in Table 6.1.

When considering the planar graph results, each of the heuristics performs best for the different performance measures. For the fraction of runs a proper colouring is achieved, the VCS and VMS heuristics perform best, with ψ taking values very close to one. The most surprising result is the poor performance of the VPS heuristic in this measure. This is because although the network is kept 'fresh' in the VPS case, there is no 'uphill' element, with the overall level of interference never increasing after a device checks itself.

Heuristic 7 VMS heuristic for node i; variable rate, mixed resolution, stochastic choice; check rate λ=1+conf(i)

Action:

```
if clock_time = τ then
    col_neigh ← {colour of neighbours}
    if col_neigh = ∅ then
        c_i ← 0
    else
        change ← false
        cur_conf ← number_of_conflicts(c_i, col_neigh)
        for col ∈ colours\{c_i} do
            conflicts ← number_of_conflicts(col, col_neigh)
            if conflicts = 0 then
                c_i ← col
                break for
            else if conflicts < cur_conf then
                c_i ← col
                cur_conf ← conflicts
                change ← true
        if not change and cur_conf > 0 then
            c_i ← uniform_discrete[0, C−1]
    τ ← τ+expovariate(1+number_of_conflicts(c_i, col_neigh))
```

When interference in the network is considered the performance of the heuristics is reversed. Now it is the VPS performing better than the VMS and VCS heuristics. This is because conflicts are occasionally added to the network is the VMS and VCS heuristics, so κ_t takes a higher value at this point in time and thus slightly inflates the average of κ_{20-200}. However, with all the heuristics returning almost negligible values for κ, this measure can be ignored for differentiating between the heuristics. This is because the network will always be resolved to an almost optimal state.

For the final performance measure, ρ, it is the VMS heuristic that again performs best. This is most notable in how efficiently proper colourings are found, with the checks over the run time returning similar levels when comparing heuristics at the same check rate.

The variable rates again here show that they are beneficial when considering ρ and therefore the efficiency of a network. Initially ignoring the λ=conf(i) check rate it is noted that the variable rates reduce the value of ρ by roughly 10 for all of the heuristics, with the λ=exp(conf(i)) check rate generally returning the fewest checks.

		ψ	ρ	time	ρ_∞	κ_{20-200}
λ=1	FCS	0.971	45.441	45.441	200.020	0.08786
	FPS	0.439	47.617	47.658	199.943	0.01312
	FMS	1	29.986	29.981	-	-
λ=1+conf(i)	VCS	0.995	39.021	37.071	211.186	0.06632
	VPS	0.463	43.150	42.352	203.526	0.01304
	VMS	1	20.339	19.547	-	-
λ=conf(i)	VCS	0	-	-	20.135	0.17158
	VPS	0	-	-	11.714	0.09037
	VMS	0	-	-	12.987	0.10873
λ=exp(conf(i))	VCS	0.982	31.935	29.154	215.175	0.01966
	VPS	0.468	36.280	35.182	205.903	0.01271
	VMS	0.998	19.489	18.319	210.655	0.01324

Table 6.1: Data from simulations on planar graphs with 200 nodes, $PL(200)$; 1000 simulations with a run time of 200 for each heuristic; ψ shows the fraction of runs for which a proper colouring was found; ρ displays the average number of relative checks used when a proper colouring was found; t represents the average run time when a proper colouring was found; ρ_∞ is the number of checks when a proper colouring was not achieved; κ_{20-200} is the average level of interference in the network between t=20 and 200.

When a network is not fully optimized this performance is reversed, now with the fixed rate heuristic using fewer checks. However, it is again the λ=conf(i) check rate that outperforms the others. For the VCS heuristic it requires 10 times fewer relative checks, whereas for the VPS and VMS heuristics this reduction value is closer to 20.

The log-scaled plots help provides a much clearer distinction between the VCS and VMS heuristics. The steeper gradient for the VMS shows that the proper colourings were found at a quicker rate than the VCS. Excluding λ=conf(i), the VCS shows a similar shape for all rates, with the two variable rates displaying steeper gradients and quicker resolutions initially. For the VMS heuristics, the blue lines initially have a constant gradient and this is particularly clear in the λ=1+conf(i) case. This signifies that the graphs were properly coloured at an exponential rate.

With the FCS and VCS heuristics being tested on both the planar and street graphs, comparisons can be made between the two. It is clear to see that the simplified structure of the street graphs with all nodes having a degree of 2, 3 or 4 makes finding a proper colouring easier and this can be seen across the majority of the performance

measures. For ψ the heuristics were able to obtain a value of almost one for both street and planar graphs, showing that the heuristics are very effective for these graph classes. When looking at relative checks, ρ, the planar graphs had an average of approximately 35 across all rates (excluding λ=conf(i)), whereas ρ was 7 times smaller for street graphs. This clearly shows that street graphs are easier to fully optimize and this is emphasized by noting that the street graphs tested had approximately 440 nodes on average, compared to 200 nodes in the planar graphs.

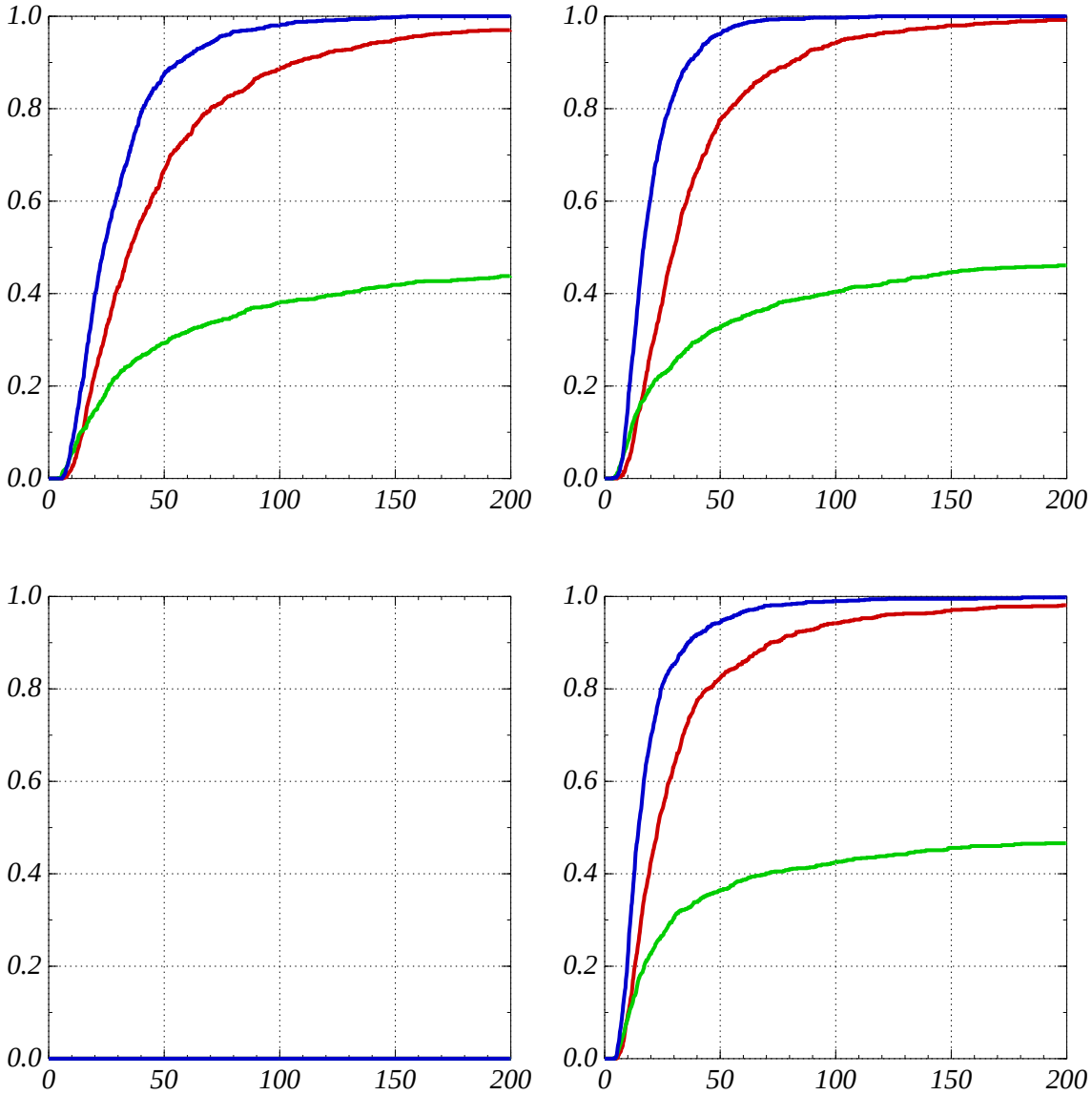


Figure 6.1: Fraction of runs for which a proper colouring was obtained for planar graphs with 200 nodes, $PL(200)$; 1000 simulations; run time of 200; horizontal axis: relative checks (ρ); vertical axis: fraction of runs for which a proper colourings was obtained (ψ); top left: check rate (λ)=1; top right: λ=1+conf(i); bottom left: λ=conf(i); bottom right: λ=exp(conf(i)); red: VCS; green: VPS; blue: VMS.

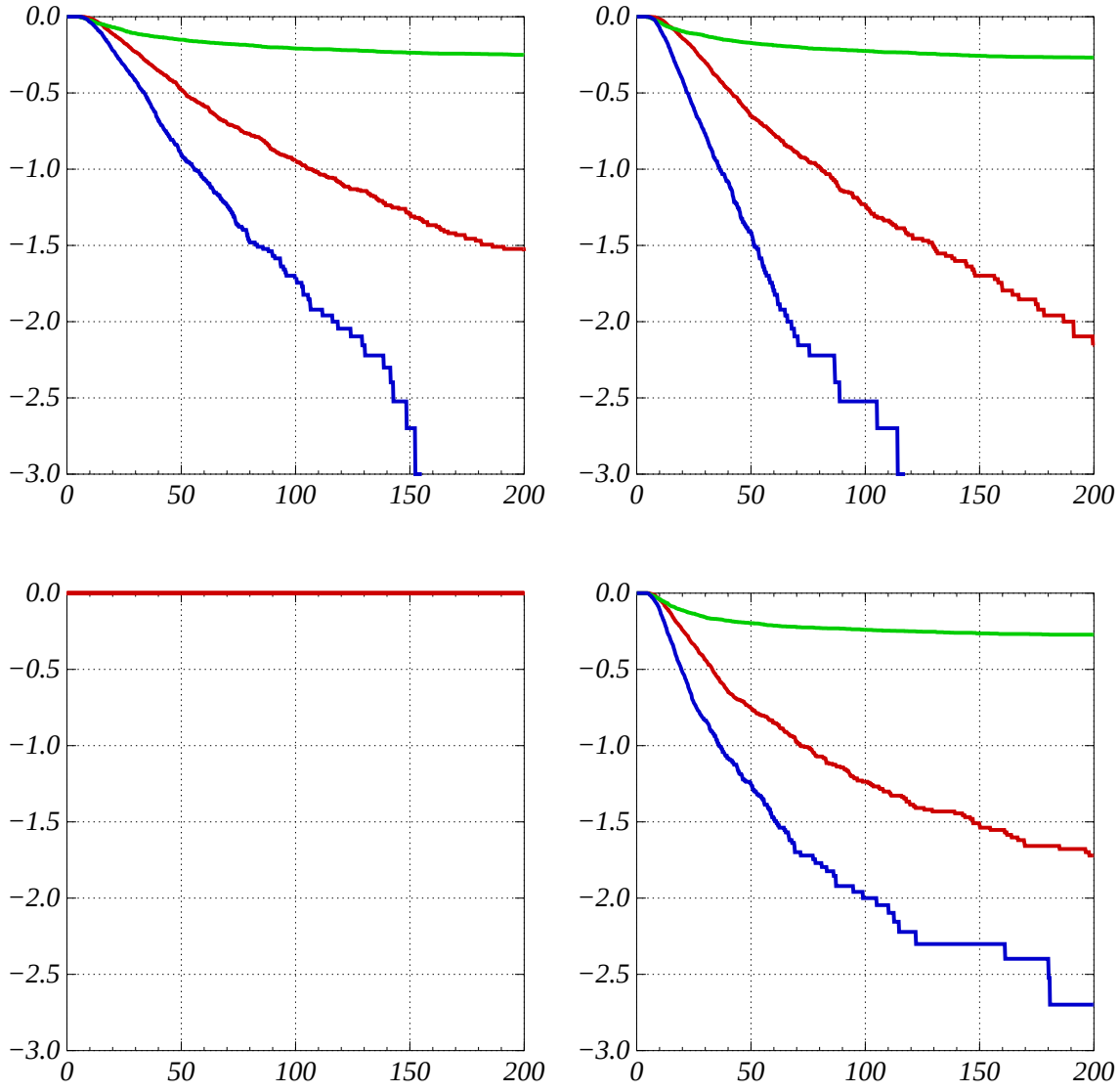


Figure 6.2: Fraction of runs for which a proper colouring was obtained for planar graphs with 200 nodes, $PL(200)$; 1000 simulations; run time of 200; horizontal axis: relative checks (ρ); vertical axis: fraction of runs for which a proper colourings was obtained (ψ) using $\log_{10}(1-\psi)$ scale; top left: check rate (λ)=1; top right: λ=1+conf(i); bottom left: λ=conf(i); bottom right: λ=exp(conf(i)); red: VCS green: VPS; blue: VMS.

Chapter 7

Martlesham Heath

In this section a real-life network of wireless devices is considered, with the data taken from the Martlesham Heath area. Here devices are considered to be BT home hubs and these have 12 channels to use. The network is then constructed in a similar fashion to a geometric random graph, but different transmission power levels, r, are used for each device. The values used are taken from a Gaussian distribution. This was to allow for signal strength to be affected by varying wall thickness in buildings. An example construction is shown in Figure 7.1 with mean 25 and standard deviation 10. Due to varying r, there will be occasions when device A will be in device B's neighbourhood, but the opposite not true. When this happens an edge will still be added, with the assumption that both devices can cause interference to each other.

To collect the position of each home hub a device to pick up the signal strength of any transmitting home hub was driven around the Martlesham Heath area and the location with the strongest reading was recorded for each home hub. Initially this meant that the position of each home hub was recorded in the middle of the road. These were then combined with the known location of all the houses in Martlesham Heath such that each home hub was assigned a house with each house having no more than one home hub attached to it. This process was done by taking the cubed distance between each recorded home hub location and every house. These were then optimized so that the shortest total distance was achieved. The cubed distance was used to heavy penalize long distances and also as signal strength drops off at a rate approximately equal to the inverse of the distance cubed.

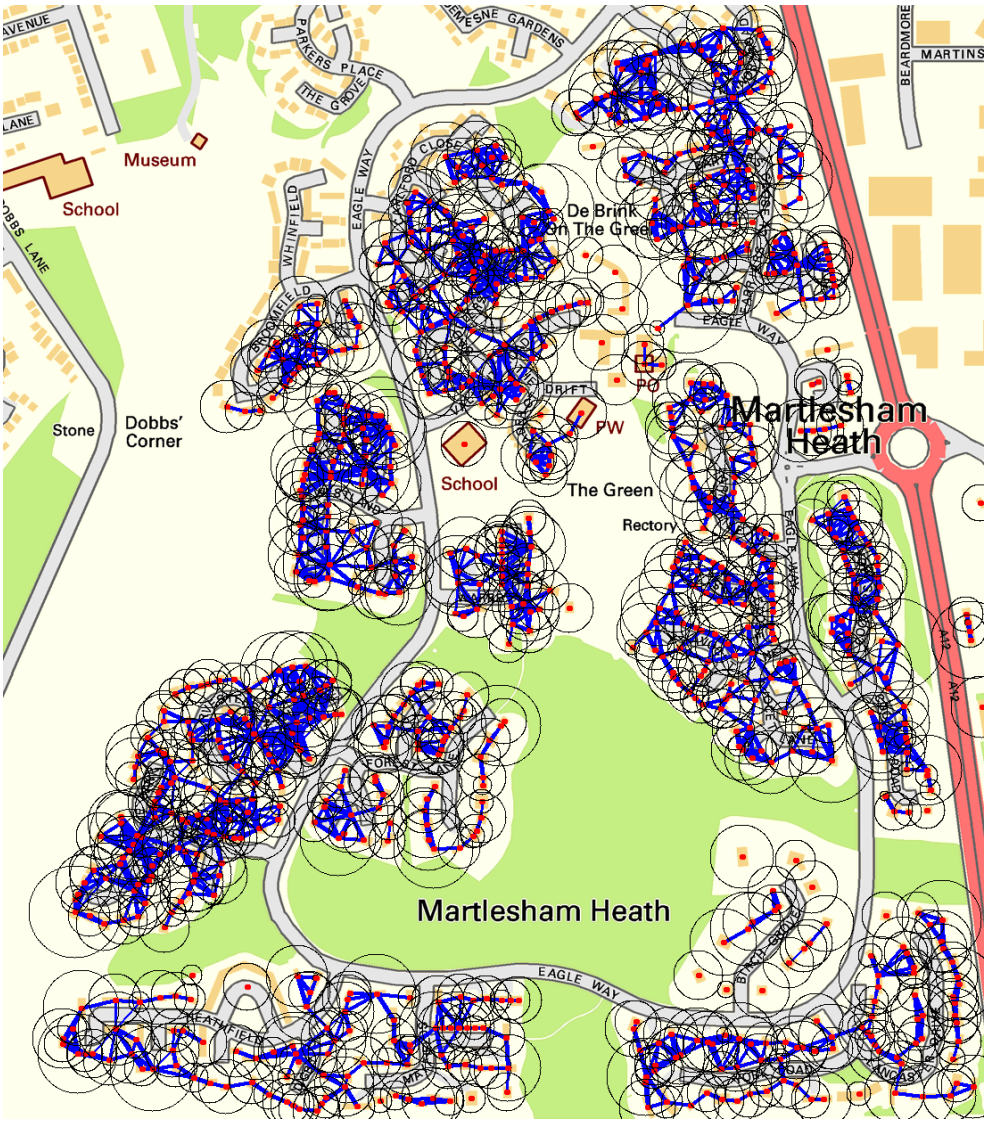


Figure 7.1: Distribution of devices in Martlesham Heath with transmission power level (r) based on a Gaussian distribution with mean 25 and standard deviation 10 (in metres).

The simulations considered relax several of the assumptions currently being used to try and add more realism to the model and to test the heuristics further. The first area considered is how channels interfere with each other. Currently there is only interference if two neighbouring devices are on the same channel. In reality if two devices are on channels close to each other (ie channels 0 and 1), then a level of interference will be present. To model this a simple linear formula will be used, equation 7.1, with the assumption that channels interfere with their five closest neighbours in the channel spectrum, with the linear example shown in Figure 7.2.

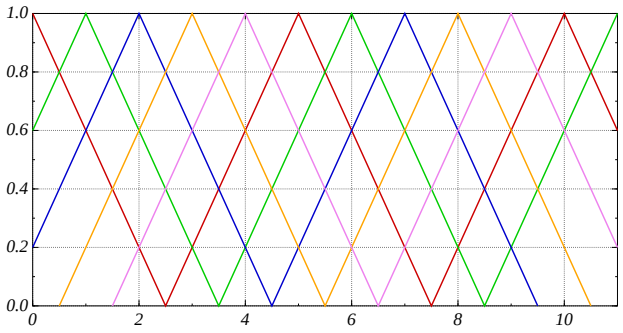


Figure 7.2: Interference between channels using a linear scale with a range of 5; horizontal axis: channel; vertical axis: level of interference; red: channels 0, 5, 10; green: 1, 6, 11; blue: 2, 7; orange: 3, 8; purple: 4, 9.

$$\text{conflict function}(c_i, c_j) = \max(1 - \frac{1}{5}|c_i - c_j|, 0) \tag{7.1}$$

With this new assumption the random colourer now has an expected value of κ_∞=4.5, so this is the value the heuristics are expected to beat. The *number_of_conflicts* function previously shown in Algorithm 2 also changes, with the new code shown in Algorithm 8.

Algorithm 8 number_of_conflicts(col, col_neigh)

```
conflicts ← 0
for j ∈ col_neigh do
    conflicts ← conflicts+conflict_function(col, j)

return conflicts
```

The second assumption to be relaxed is the constant graph process, with variable graph processes being introduced. To do this when the simulations are initialized, a proportion of devices will start off, meaning that they will cause no interference to their neighbours. As the simulation progresses with time, devices will then turn on and off at independent points in time. When a device turns off any conflicts it currently has with its neighbours will be removed from κ, but neighbours will not be aware the interference has been removed until they next check themselves. When a device is turned on and becomes active it goes through the initialization process shown in Algorithm 1.

A variety of heuristics and rates will be tested on the network from Martlesham Heath, with the aim to highlight their strengths and weaknesses. The first heuristic

will be the FCD with a check rate of λ=1. With this being the simplest heuristic, it is expected to return the worst results, but will help demonstrate how the improvements discussed in this report have helped performance. The second heuristics will be VMS and VPD with λ=conf(i). It is anticipated that these will provide acceptable interference levels but excel when considering the efficiency of the networks. The final heuristic to be tested will be VPS with λ=exp(conf(i)). So far this has proved to be the most effective heuristic at finding optimal resolutions and would expect it to return the lowest interference levels. A random colourer will also be tested, with devices uniformly being assigned a new channel every time they are switched on.

The networks consisted of 920 devices, with a mean degree of approximately 4.5 using the r values stated before. 1000 simulations were run for each heuristic with a run time of 1000. The networks were initialized with a quarter of devices switched off at time t=0. The rate at which devices turn on and off is set to 5 per unit time. This means on average five devices will switch off and five devices will be switched on during each time step. These devices will be independently chosen. The performance of the heuristics is displayed in Figures 7.3 and 7.4. Figure 7.3 displays δ, the average level of interference each device currently active has at time t, while Figure 7.4 shows the number of checks, ρ, in the network as time progresses.

The first result that stands out in Figure 7.3 is that the FCD heuristic performs worse than the random colourer, which would not be expected as there is no optimization occurring with the random colourer. However, this is due to the greedy nature of devices operating with the FCD heuristic. This is because a device will only move to a channel completely free of interference, so with the linear channel model being used there is very little scope for optimization. Often devices will be carrying out the check function and finding channels that will have almost negligible levels of interference, but not zero, so no change is made. This means that devices will generally only utilize a few non-overlapping channels, so 0, 5 and 10 for example.

The second result that stands out is the VPD heuristic returning less interference per device than the VMS heuristic for the λ=conf(i) check rate. This is down to the stochastic choice element, so devices are gaining conflicts without their knowledge so not checking to resolve this. The changing of the conflicts does not occur in the deterministic choice heuristic, which allows for the lower level of interference to be maintained, even though the network is more liable to become stuck in a sub-optimal state. The VPD heuristic example is much smoother with more erratic behaviour being demonstrated by the VMS heuristic and this is again due to the reason mentioned above.

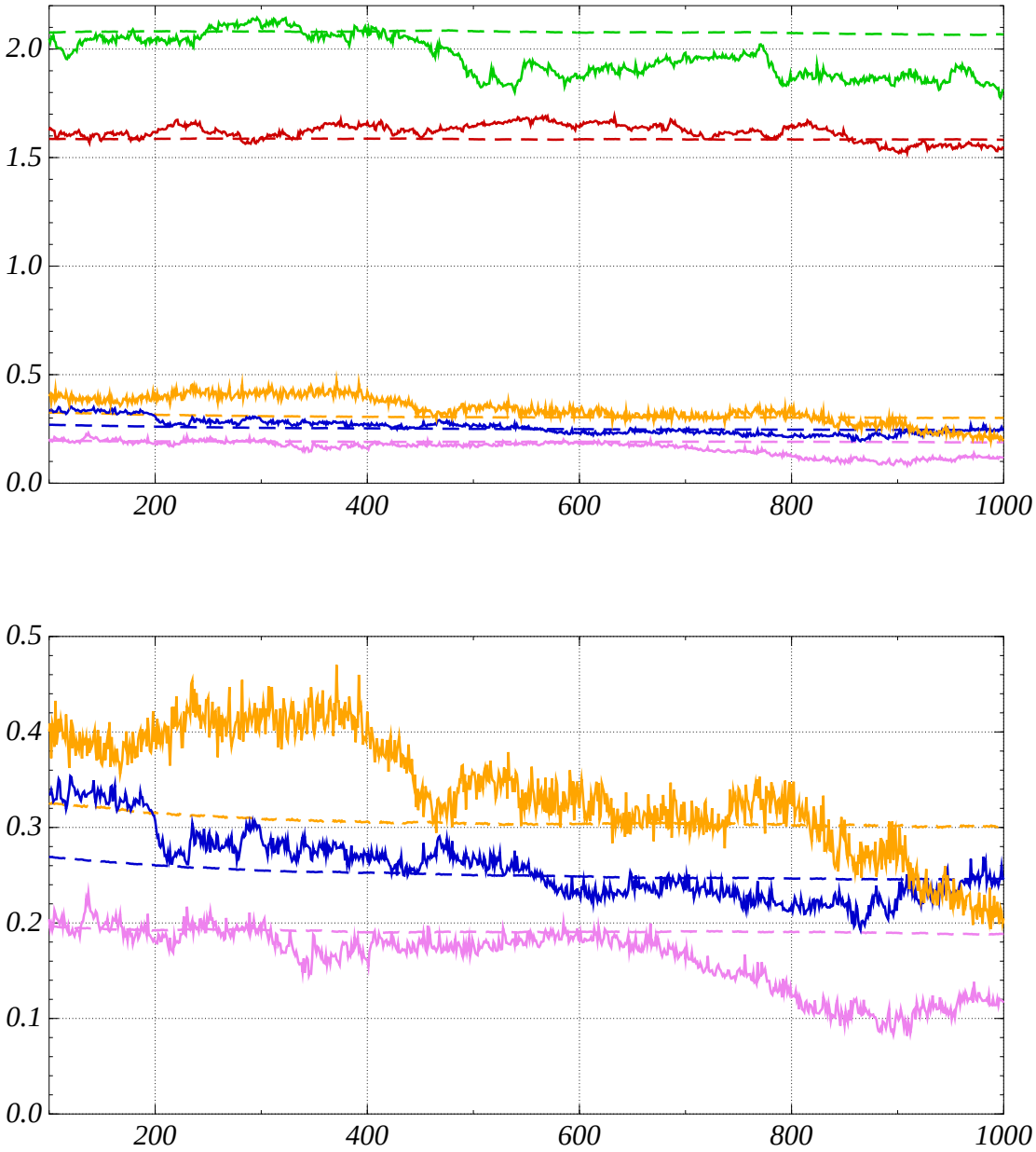


Figure 7.3: Average level of interference per device with devices switching on and off and a linear channel interference model used with a range of 5; horizontal axis: run time; vertical axis: conflicts per device (δ); red: random colourer; green: FCD with check rate λ=1; blue: VPD with λ=conf(i); orange: VMS with λ=conf(i); violet: VPS with λ=exp(conf(i)).

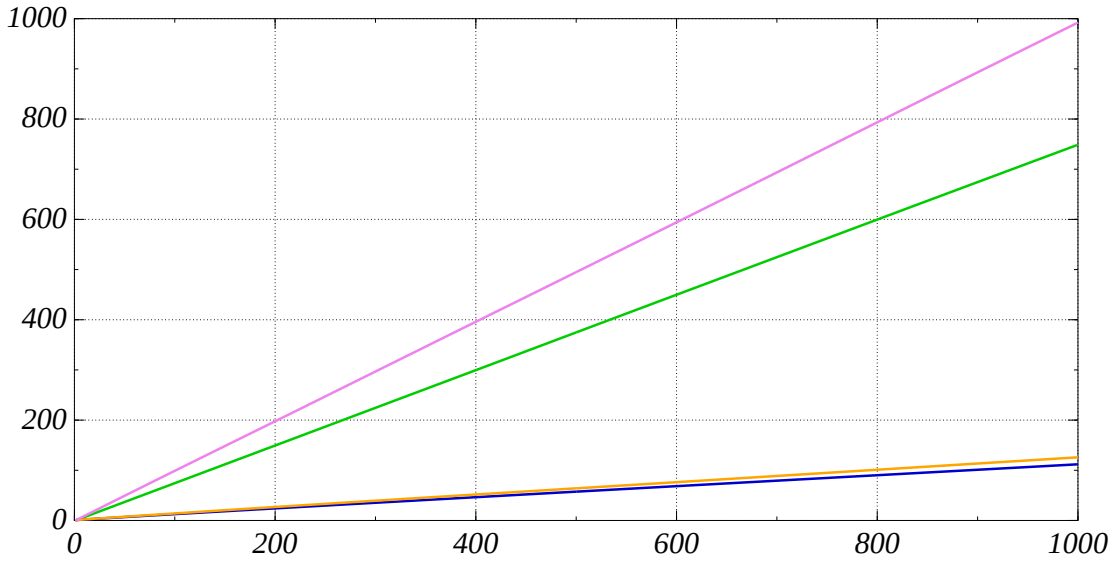


Figure 7.4: Relative checks in a network with devices switching on and off and a linear channel interference model used with a range of 5; horizontal axis: run time; vertical axis: relative checks (ρ); green: FCD with check rate λ=1; blue: VPD with λ=conf(i); orange: VMS with λ=conf(i); violet: VPS with λ=exp(conf(i)).

The lowest level of interference per device was achieved with the VPS heuristic while using a variable check rate of λ=exp(conf(i)), which is as the predictions anticipated. When averaged over all the runs this heuristic returned a value of δ approximately 8 times smaller than the random colourer, showing this heuristic proves very beneficial when trying to optimize the network. The example run also displays that the interference level remains relatively smooth throughout.

When viewing the number of relative checks in Figure 7.4, the variable rates when devices do not check themselves after they have completely resolved their conflicts optimized the networks in the most efficient way. Here the λ=conf(i) was 10 times more efficient than the exponential rate. The random colourer is not included on the plot because no checks are allowed to occur as no optimization of the network takes place. Therefore for the heuristics to be beneficial they cannot take up too much of a device's power by checking for conflicts.

For users the main concern is the throughput of their device, meaning the speed with which it is able to send and receive information. Small differences in the level of interference a device is receiving often has no impact on this speed. This is because each device will usually have 4 or 5 different levels of throughput available to use, with each of these covering a range of interference levels. Therefore although the

exponential rate returns the lowest level of interference per device, in reality each device will have the same throughput as the the λ=conf(i) rates, so the user would feel no benefit. This implies the λ=conf(i) rate has an overall better performance as it enables devices to conserve power.

Chapter 8

Conclusion

In this report two main areas are considered and different recommendations can be made as to which of the local distributed asynchronous heuristics work best. The first is a more theoretical one where the only measure of interest is trying to obtain a proper colouring for a graph. The second is reducing the number of conflicts in a graph when speed and efficiency are important.

When attempting to obtain a proper colouring the best heuristic is variable rate, mixed resolution with stochastic choice. The key feature of the rate is that devices keep checking themselves even if they completely resolve their conflicts and the exponential rate enabled proper colourings to be achieved at the fastest rate by introducing 'targeting' so areas of high conflict are resolved first. The stochastic choice is also very important with graphs been able to go in an 'uphill' direction which prevents solutions stabilizing at a sub-optimal state. This can also be seen in devices willing to behave in an altruistic fashion as they reduce their own performance in the short run in an attempt to help achieve an optimal solution in the long run.

When the application to wireless networks is considered, the efficiency with which a network optimizes itself becomes very important and a trade-off between this and interference levels has to be considered. This is because every time a device checks for conflicts it uses up power, so reducing the number of checks implies longer battery life. For this the variable rate, partial resolution only, deterministic choice heuristic proved best. The key feature this time is the variable rate, where devices should stop checking for conflicts once they reach a zero-conflict state. Although very few optimal solutions are achieved with this heuristic, networks are consistently reduced to acceptable levels of interference which would not effect the performance and this was achieved significantly more efficiently than for other heuristics.

Bibliography

[1] http://www.stanford.edu/group/stanfordbirds/text/essays/Brood_Reduction.html.

[2] http://keithbriggs.info/graph_theory_and_W.html.

[3] http://docs.python.org/library/random.html.

[4] http://keithbriggs.info/very_nauty.html.

[5] http://www.shutterstock.com.

[6] D. Achlioptas and A. Naor. The two possible values of the chromatic number of a random graph. *Annals of Mathematics*, 162(3):1333–1349, 2005.

[7] H. Ayanegui and A. Chavez-Aragon. A complete algorithm to solve the graph-coloring problem. In *Latin American Workshop on Non-Monotonic Reasoning 2009*, pages 107–117, 2009.

[8] V. C. Barbosa. *An Introduction to Distributed Algorithms*. Massachusetts Institute of Technology, 1996.

[9] A. Denise, M. Vasconcellos, and D. J. A. Welsh. The random planar graph. *Congressus Numerantium*, 113:61–79, 1996.

[10] M. Dorigo, V. Maniezzo, and A. Colorni. The ant system: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics - Part B, Vol.26, No.1, 1996, pp.1–13*, 1996.

[11] P. Erdős and A. Rényi. On random graphs. i. 1959. Publ. Math. Debrecen, Vol. 6 (1959), pp. 290–297.

[12] S. Fitzpatrick and L. Meertens. Soft, real-time, distribusted graph colouring using decentralized, synchronous, stochastic, iterative-repair, anytime algorithms. Technical report, Kestral Institute Technical Report KES.U.01.05, May 2001.

[13] S. J. Pumphrey. Solving the satisfiability problem using message-passing techniques, May 2001.

[14] L. Song. *Random graph models for wireless communication networks*. PhD thesis, Queen Mary University of London, March 2010.

[15] G. Tel. *Introduction to Distributed Algorithms*. Cambridge University Press, second edition, 2000.