Digital Business Ecosystem

Contract number° 507953

Workpackage11
Dynamics of networks

Deliverable 11.1
Report on The Flow of Software Components in a Static Network

Information Society Technologies

Project funded by the European Community under the "Information Society Technology" Programme

Contract number: 507953
Project acronym: DBE
Title: Digital Business Ecosystem

Deliverable Nº: D11.1
Due date: 30/04/2005
Delivery date: 30/04/2005 (prospective)

Short description:
This report describes our studies of how software components may flow in though a static network, and how such a flow may influence the progress of evolution of higher-level software. It comprises two main contributions. The first is a mathematical framework for analysing how effectively information can flow stochastically through a network. We are able to analyse some network topologies, and provide empirical results for others. The second contribution is an empirical study of how effectively evolution can be used to solve the set-cover problem (an abstraction of the problem faced by the DBE).

Partners owning: UBHAM (Jonathan E. Rowe, Boris Mitavskiy, Kennon Ballou)
Partners contributed: STU, LSE, ICL, SUN
Made available to: DBE Consortium and European Commission

Versioning		
Version	**Date**	**Author, Organisation**
1.0	22/04/2005	UBHAM
0.9	01/04/2005	UBHAM

Quality check
1st internal reviewer: Miguel Vidal (SUN)
2nd internal reviewer: Philippe De Wilde (ICL)

Contents

Executive Summary

This report describes our studies of how software components may flow through a static network, and how such a flow may influence the progress of evolution of higher-level software. It comprises two main contributions. The first is a mathematical framework for analysing how effectively information can flow stochastically through a network. An important point is that information is actually copied from site to site, rather than moved. Starting with some information at an initial site, we seek the expected time until the whole network has received the information. There is an analogy with the spread of infection through a network of acquaintances in epidemiology. We are able to analyse some network topologies, and provide empirical results for others. The mathematical analysis is continuing research. The second contribution is an empirical study of how effectively evolution can be used to solve the set-cover problem (an abstraction of the problem faced by the DBE). We look at how different topologies affect the time taken to solve the problem and, importantly, the effects on the diversity maintained within each habitat.

Work planned for the second phase of the project includes:

- completing the theoretical analysis of propagation time for different network topologies,
- studying the effects of hierarchical structuring in networks on propagation dynamics,
- using clustering algorithms to examine this kind structure.

The "customers" of this report are primarily the Science partners, and those Computing partners interested in the impact of network topologies on system performance. In particular:

- ICL — the effects of changing network topology on efficiency
- SUN — implications for the growth of the DBE networks
- LSE — implications for the growth of social networks amongst DBE users.

Chapter 1

Introduction

Despite being the ”backbone” of the European economy, most Small-to-Medium Enterprises (SMEs) have failed to take full advantage of the technology resources available to them [31]. In particular, most are not ready to use the Internet (or other technology) as a business tool. The reasons are widely understood: lack of resources, few skilled employees, technology poorly adapted to SMEs, and especially a lack of awareness of the potential benefits [31]. In addition, there are regional differences in technology adoption, with SMEs in Nordic and some western European areas being much more likely to take up new technologies than SMEs in eastern or southern Europe.

1.1 The Digital Business Ecosystems Project

To address this problem, the European Commission has funded several projects designed to encourage cooperation and technology adoption by SMEs. One such project, the Digital Business Ecoystem (DBE), aims to develop paradigms and technologies that will allow businesses to work together in a nature-inspired, organic fashion. This will hopefully allow SMEs to adopt new technologies faster and more successfully.

Although the concept of businesses working together like an ecosystem is not a new one ([35], [34], [38]), the DBE project focuses on the *digital* aspect of business cooperation, in particular software components, services, applications, and business models. The goal is to provide a digital framework for these components to act as ”digital species” that will interact, reproduce, and evolve based on market forces. This evolution in particular is of interest, since it should encourage the growth of new and unforeseen components that are inherently adapted to the surrounding environment.

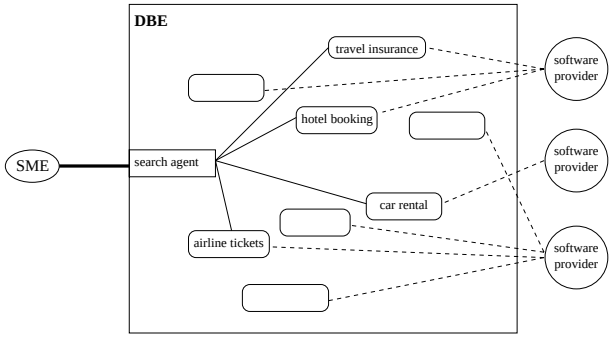


Figure 1.1: DBE Framework.

Software Specifications

For most SMEs today, choosing the right software to use for its business processes can be a daunting task. With a huge number of choices for most applications, a typical SME usually does not have the competency or experience to choose the software that is right for them. When software is chosen, it is rarely tailor-made for the SME, and requires significant investment of time and money (either in house or through expensive outside contractors) to custom-fit it to the SME's business process.

To cope with this problem, the DBE proposes to create a "Business Modelling Language" (BML), which will allow SMEs to detail their software needs in the form of a software requirement. In the functioning ecosystem, software producers will create modular pieces of software and release them into the framework; intelligent search agents (ISAs) will use the software requirements as a blueprint and interact with the software modules in the ecosystem to assemble a collection of modules that together meets the needs of the SME.

For instance, an SME wanting to offer holiday travel packages would probably need software that enables on-line air ticketing, hotel reservations, car rentals, travel insurance, etc. Instead of having to create all of these features themselves, the SME could simply create a BML script that details all of these components as a requirement, and hand it to an ISA. Each of these modules might be produced by a different software provider; the ISA will find the optimal combination to give the maximum cost benefit to SME (see Figure 1.1). In Chapter 5 we give a formal description of a simplification of this problem, which is a slight modification of the Set Covering Problem.

One of the important research areas in the DBE project is to determine the ideal way for all of these SMEs to interact. In particular, since computer networks (and the Internet especially) are the standard paradigm for computing in the present, it seems likely that some sort of distributed approach will be used for the DBE. However, there is a very wide range of possible topologies in which this network might be formed, and thus it is very important to understand the effects that

different network topologies may have on the evolution and growth of the DBE.

The first part of the research presented here is a mathematical and empirical study of a basic question: if some information (or software service) is created at one node of a network, and is transmitted along connections (with a certain probability of success), how long will it take before all nodes in the network have received a copy? The answer clearly depends on the number of nodes, the transmission probability and the network topology. Such a natural question turns out to be rather difficult to analyse in practice. We give a range of asymptotic results and show the results of some empirical experiments. This work can be applied directly to the question of network structure in the DBE, since there are a number of possible ways in which it might grow. For example, nodes may be allowed to connect randomly, or may be required to connect to a hub. Or there may be a stricter distribution structure (a grid or tree structure) imposed by the centre. Different topologies have different associated propagation times, which will become significant as the DBE network gains more and more nodes. It is imperative that this relationship is understood and taken into account in the DBE network design (e.g. by SUN).

Such results also have implications at the "social" level, which is also of interest to DBE researchers (e.g. LSE) as they consider how effectively users share information through their connections within the DBE and outside of it. Here the qualitative nature of the results will be more significant than concrete figures, and it will prove interesting to relate this technical piece of mathematics to a social modelling problem.

In the second part of the research, we developed a genetic algorithm model of an ISA, since not only are genetic algorithms nature-inspired (which fits with the aims of the DBE project), but they also have been shown to be successful at search tasks such as this [7]. One approach that has been tried with GAs is to parallelise them in an attempt to reduce the time it takes to find good solutions. Because of its use of multiple populations and similarity to a possible model of the DBE, we have also developed a parallel genetic algorithm model with arbitrary topology in order to examine the effects of changing topology on the flow of genetic information. We also present here NetEvo, a software simulation package that implements this parallel genetic algorithm model.

The organisation of this report is as follows. The second chapter will discuss several different background areas that are applicable to the project. Chapter 3 will present a mathematical analysis of how software (or information) can propagate through a static network. The emphasis is on asymptotic bounds as the size of the network grows. Chapter 4 presents empirical studies of the same phenomena. The fifth and sixth chapters will introduce NetEvo, our simulation tool, and the seventh will detail our experiments and results.

Chapter 2

Background

This chapter will introduce some of the concepts necessary to understand this project, as well as some closely related research areas that are of interest. The first section gives a brief introduction to graph theory, while the second and third introduce Genetic Algorithms and Parallel Genetic Algorithms respectively. The fourth section describes the Set Covering Problem. The fifth section discusses the field of epidemiology and the final section describes the theory of metapopulations.

2.1 Graphs

In order to formally analyse the properties of networks, it is necessary to define a type of object called a *graph*. This section follows Watts' [40] overview of Wilson and Watkins' work on graph theory [41].

Formally, a graph consists of a non-empty set of items called *vertices*, and an unordered list of pairs of vertices called *edges*. If an edge contains vertices a and b, those vertices are said to be *connected* [41]. Sometimes vertices are informally called *nodes*, and the two words will be interchangeable here. Although there are other types of graphs that put a weighting on edges or where the direction of the edge matters (called *weighted* and *directed* graphs, respectively), this paper will focus on undirected, unweighted graphs. If it is possible to move from one vertex in a graph to any other vertex in a finite number of edge moves, that graph is said to be *connected* [40]. A *fully connected* graph is one in which every vertex is connected to every other vertex. It should follow that for an undirected graph of size n, a fully connected graph will contain $\frac{n(n-1)}{2}$ edges. A *sparse* graph is one in which the number of edges is significantly less than this maximum.

Degree

The number of edges that are connected to a given vertex is said to be the *degree* of that vertex. One important characteristic of a graph is the *average degree*, since this gives a relationship between the size of the graph and its sparseness [40]. The real-world ramification of a graph with a high average degree is that individual vertices will have a very large number of neighbours, which would place a higher burden on communication.

Characteristic Path Length

Perhaps the most important statistic of graphs in terms of this project is the *characteristic path length*, which is the average distance between any two vertices. Distance is measured by the smallest number of steps across edges from the first vertex to the second. This statistic is important because it will determine how long on average information will take to diffuse across the network.

Clustering

When there is a subset of vertices in a graph that are fully connected to each other, this is called a *cluster*. The *clustering coefficient* of a graph is the probability that the neighbours of a particular vertex will also be connected to each other [40]. A graph with a high clustering coefficient implies a large number of clusters, while a low clustering coefficient implies disconnected vertices.

2.2 Genetic Algorithms

Genetic algorithms (GAs), originally formulated by Holland ([23], [24]), are probabilistic optimisation techniques inspired by evolution. In nature, organisms that are more adapted to their environment will survive longer, giving them a higher chance of propagating their genetic material which in turn will create new organisms with a higher chance of survival. This concept of *natural selection*, discovered by Charles Darwin [12], ensures that over successive generations organisms will become more and more adapted to their environment, i.e., more ”fit”.

GAs harness this power by simulating the process of natural selection. Potential solutions to the optimisation problem are encoded into a special form (which is called a *genome*); a population of these genomes is then initialised randomly. Each generation, a sequence of *genetic operators* are applied, creating a new generation which is (hopefully) better than its parents.

Representation

The genetic representation of the potential solutions is important since this determines the space in which the GA will be able to search. Additionally, some representations will be easier to work with than others. In addition to the encoding, it is important to have a logical mapping from the genomes back to the potential solutions, since the GA actually functions only on the encoded genomes. The process of mapping a genome back to a potential solution is called *expression*.

Bit string encodings are a popular choice for GAs since they are very easy to work with and often have a very intuitive mapping to the potential solution. With this encoding, each chromosome is a bit string, and each bit corresponds to a gene.

Fitness

Standard GAs are ”black box” algorithms; in other words, the algorithm only requires an objective function to give a relative fitness value to the expressed genome [15]. The actual problem being solved could literally be anything; the GA does not care. Depending on the genetic operators used, typically each generation each genome is expressed into the potential solution and then a fitness value is assigned to the genome based on how good the solution is.

Occasionally, due to genetic operations, an infeasible individual can be created. There are several ways of dealing with this: the infeasible individual can simply be ignored or regenerated; the GA can attempt to repair the individual to make it feasible; or a penalty to the fitness can be applied. Each of these approaches has benefits and drawbacks.

Genetic Operators

The two most common genetic operators are *crossover* and *mutation*. Crossover models sexual reproduction by taking multiple (usually two) individuals as parents and creating a number of new individuals as offspring [29]. With a bit string representation, this usually involves creating a new bit string that is made up of bits from the parents. Two popular methods are single point and uniform crossover. Single point crossover randomly selects a position in the bit strings and selects the bits up until that position from the first parent and then the rest of the bits after that position from the second parent, creating two new offspring (see Figure 2.1).

Uniform crossover iterates through all bit positions and randomly decides to select the bit from the first parent or the second parent; this can allow a bias towards one parent or the other, perhaps based on relative fitness [28].

There are many different ways of selecting which individuals will be crossed over; two popular methods are roulette wheel selection and tournament selection. Roulette wheel selection gives a relative probability to each individual based on its fitness, and selects parents based on those probabilities. While this ensures that more fit genomes have a higher chance of mating, this

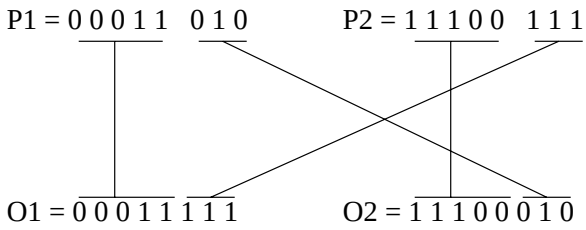


Figure 2.1: Single point crossover. Position 5 is selected as the crossover point and two offspring bit strings are generated from the parent bitstrings.

also requires absolute fitness values [15]. Tournament selection, on the other hand, only requires a relative measure between two individuals; two (or more) individuals are selected at random, and the individual with the highest fitness is selected [28]. When two individuals are selected, this is called *binary tournament selection*, and is a popular choice.

The mutation operator is used to create diversity in the population. Although there are also many different ways of doing this, a popular method is to randomly select individuals from the population and then flip each bit with a certain probability. Another method swaps two (or more) bits with each other. It is tricky to know how high to set the mutation rate; if it is too high, it can create many bad solutions in the population, but if it is too low then it is ineffectual. For this reason, some GAs use an adaptive mutation rate which changes over time based on the current diversity of the population [29].

2.3 Parallel Genetic Algorithms

Parallel GAs attempt to increase the speed of a standard GA by dividing the task into parts and evaluating these parts simultaneously [8]. Especially when using multiple processors, this can decrease the time required to find a good solution dramatically. This section will give a brief overview of PGAs, following closely Cantu-Paz's excellent survey of the topic [8], and the interested reader is encouraged to look there for a more thorough review.

Cantu-paz classifies PGAs into three main types: a) global single population master-slave GAs, b) single population fine-grained GAs, and c) multiple population coarse grained GAs.

Master-Slave PGAs

Master-slave PGAs are characterised by having a single population, as in a simple GA, but the fitness evaluation is run in parallel across multiple processors. Typically, the "master" processor stores the population and when it is time for the fitness evaluation it dynamically assigns each individual to a "slave" processor (see Figure 2.2). Since only the fitness evaluation is run in parallel, in other respects typical master-slave PGAs function identically to regular GAs. This

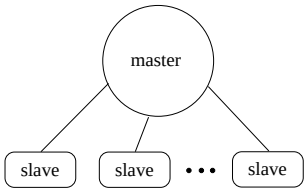


Figure 2.2: Master-slave PGA. The master processor sends individuals to slave processors to be evaluated for fitness.

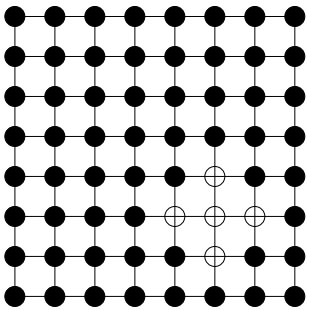


Figure 2.3: Fine-grained PGA. This population is structured into a 2D grid, and genetic operators are only applied to a spatial neighbourhood.

type of PGA is especially suited to situations in which the fitness evaluation time dominates the algorithm; another advantage is that since they are algorithmically identical to regular GAs, all the existing theory is directly applicable [8].

Fine-grained PGAs

Fine-grained PGAs use a spatially-structured population, where the genetic operators are confined to a small spatial area [8]. These ”neighbourhoods” are connected, allowing good solutions to propagate throughout the population. Typically, fine-grained PGAs are implemented on massively parallel computers, and the population is structured into a 2D mesh where each individual is controlled by a single processor (see Figure 2.3).

Multiple-population Coarse-grained PGAs

Multiple-population (also called ”multiple-deme”) PGAs contain multiple populations that occasionally communicate with each other by migrating individuals back and forth. Inside of a single population, the algorithm essentially functions as a standard GA, although the overall algorithm behaviour can be quite different [8].

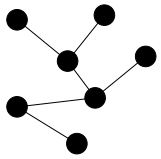


Figure 2.4: Multiple-population PGA. Multiple populations are connected into a network, and individuals occasionally migrate between the populations.

This type of PGA presents many interesting questions, such as the exact effect that migration and topology have on the overall performance. At a high enough migration rate, a multiple-population PGA can begin to perform the same as a single large population GA [8]. While most researchers study PGAs in order to find faster optimization algorithms, multiple-population PGAs do provide some interest since they much more closely model natural migration in ecosystems.

Due to its similarity with the proposed DBE framework, this is the type of model that we have developed (see Chapter 5).

2.4 Set Covering Problem

As mentioned in the introduction, the problem of matching software specifications with requirements is a slight modification of the Set Covering Problem. Before detailing the modified version, it is important to first describe the classic problem.

The Set Covering Problem (SCP) is an NP-Complete problem of finding the minimal cost subset of columns in a binary matrix in which each row in the matrix is covered by at least one column in the set, and each column has an associated cost [14]. Beasley and Chu formally define it as,

Minimise

$$\sum_{j=1}^{n} c_j x_j \tag{2.1}$$

subject to

$$\sum_{j=1}^{n} a_{ij} x_j \geq 1, i = 1...m \tag{2.2}$$

$$x_j \in \{0, 1\}, j = 1...n \tag{2.3}$$

where a_{ij} is zero-one matrix with m rows and n columns, c_j is the cost vector, and $x_j = 1$ if column j is included in the solution or $x_j = 0$ if not [7]. If all the costs are the same, it is called a *unicost* SCP.

Beasley and Chu [7] develop a GA to solve the set covering problem. They use the obvious representation for individuals, a bit string, where each position represents a column and a 1 represents that column being in the set and 0 otherwise. The fitness of an individual is simply $f_i = \sum_{j=1}^{n} c_j s_{ij}$ where s is the bit string. One ramification of using a bit string representation is that as a result of genetic operations the solutions may not be a cover (i.e. infeasible). Beasley and Chu choose to use the repair method since they have trouble determining a good penalty function [7]. They use binary tournament selection and create their own specialised crossover operator which they call *fusion*. Their GA is able to find the proven optimal solution to smaller-sized problems (which presumably were solved first by a deterministic algorithm) and high quality solutions to larger problems.

Solar, Parada, and Urrutia [37] present a master-slave PGA for solving the SCP that appears to outperform a standard GA. They also use a bit string representation and standard genetic operators. They first try to enforce a penalty function on infeasible solutions by adding a penalty based on the number of uncovered rows; however, despite getting some good results they too decide to use a repair method [37]. Multiple populations each function as a normal GA, but after each generation they transmit their best individual to the master node, which then chooses the best and replaces the worst individual in each population with the global best. They find that when a population size is divided into p populations of $P_L = \frac{P_G}{p}$ each, the parallel search takes fewer generations than the corresponding search through P_G individuals to arrive at an equivalent fitness [37].

2.5 Epidemiology

As individuals in a PGA interact and share information, this information propagates throughout the population very similarly to the way in which diseases spread through human populations. The study of the dynamics of disease spread, called *epidemiology*, may be useful in the DBE project to understand the propagation of information in the framework.

Epidemiology as a discipline is only about a century old. Along with the discoveries of bacteriology and dramatic increases in sanitation in the 19th century came the biological and social groundwork for a more theoretical study of diseases, and in the late 19th/early 20th century the first mathematical formulations of epidemiology appeared [3].

This section will present several important concepts in epidemiology, beginning with some of the fundamental deterministic equations that many epidemic models use. Then we will look at some of the differences between deterministic and stochastic equations and how each has its

advantages and disadvantages.

Directly Transmitted Diseases

At the beginning of the 20th century, W.H. Hamer [16] formulated a mathematical model for epidemics that became the base for most deterministic epidemic models since. Even though simple, it was still able to predict phenomena such as periodic epidemic cycles [3].

Conceptually, this basic model considers an epidemic as a population ecology problem, where uninfected individuals make up potential hosts and infected individuals are the population in question. With some assumptions, the behaviour over time is very similar. Specifically, for this first model we assume that the disease is directly transmitted from individual to individual and that the number of individuals in the population remains constant (which is a valid assumption in western societies [2]).

We divide the population into two categories: susceptibles, those that are normal and have not contracted the disease, and infectives, which are not only infected but are also contagious to those around them. Thus in our model, $N = S + I$, where N is the total size of the population and S and I are the corresponding size of the two categories.

The sizes of these two groups vary over time very much like the *logistic curve*, a recurring shape in population ecology, although the shape of one group's curve is the inverse of the other. If we assume that the rate at which susceptibles become infected is proportional to the number of contact pairs (the *mass action principle* [16]), we then arrive at,

$$\frac{dI}{dt} = aIS \tag{2.4}$$

where a is called the *transmission coefficient*. This transmission coefficient is roughly the probability of infection. Since we know that $N = S + I$, we can rewrite S as $S = N - I$ and replace in the previous equation to produce,

$$\frac{dI}{dt} = aI(N - I) \tag{2.5}$$

If we think of I as a proportion of N (i.e., set $N = 1$), we then arrive at the logistic equation,

$$\frac{dI}{dt} = aI(1 - I) \tag{2.6}$$

with a carrying capacity equal to 1.

We can continue to add complexity to our model by creating a new category for people, those that have recovered and are immune. With these three categories, we now have a model which is generally referred to as an SIR model [39]. We now note a new equation for dS,

$$\frac{dS}{dt} = b - aIS - wS \tag{2.7}$$

where b is the natural birth rate and w is the death rate. If we set v to be the rate at which infecteds recover, the new equation for dI is now,

$$\frac{dI}{dt} = aIS - (w + v)I \tag{2.8}$$

The equation for dR is simply,

$$\frac{dR}{dt} = vI - wR \tag{2.9}$$

These three equations are usually called an SIR system of directly transmitted diseases; occasionally a fourth state, 'E', is introduced to simulate individuals who have been exposed but are not yet infective [39]. For a more thorough discussion of these topics, the reader is encouraged to consult [2], [3], or [1].

Basic Reproductive Rate

From analysis of these equations, we can find a quantity called the *basic reproductive rate*, which is the expected number of secondary infections caused by an infected individual. This quantity is of great importance in epidemiology since it represents a threshold above which a disease will become epidemic, yet below which will eventually die off [2]. In the case of our system of equations,

$$R_0 = \frac{a}{vcolour} \tag{2.10}$$

where R_0 is the basic reproductive rate. If R_0 is greater than 1, then the disease will persist; otherwise it will eventually die out. Conceptually, this is the point at which the number of new infectives is greater than the number of recovering or dying infectives. It is interesting to note that since R_0 depends on the population size, there is a value for N below which an otherwise highly virulent disease will die out. This suggests that epidemics might sometimes have less to do with an actual disease than with current demographic trends.

Population Structure

The epidemiological formulations above assumed a population without spatial structure. However, it is important to consider the spatial distribution in order to truly understand the situation [33]. On the other end of the spectrum, some models assume a completely structured population in which each individual is treated as a vertex in a graph.

Kuperman and Abramson [26] study the dynamics of an epidemic in a population structured into a small-world graph (see Section 5.4). Using an SIRS model on the graph, they find that as the network transitions from order to disorder, there is a finite point at which an infection which had remained relatively endemic "breaks out" and spreads in oscillations quickly throughout the network. The interested reader is referred also to the related paper by Zanette [42].

In between completely structured and completely unstructured lies another model which combines some elements of both; this is the topic of the next section.

2.6 Metapopulations

The migratory aspect of individuals amongst populations in a PGA bears a strong similarity to a research area in population ecology called *metapopulations*. Metapopulation models break up a population into a collection of subpopulations, where each subpopulation behaves as a separate population, and observe the growth and/or extinction of the collection as a whole.

Levins [27] is generally credited with coining the term metapopulation, although there is some evidence that ecologists had used the concept before [19]. In any case, Levins did explicitly formulate the dynamics of a collection of populations as a whole, and his early model is the base of many of the dynamic metapopulation models in use today.

In the Levins model, as related by Hanski [17], we assume that each subpopulation (or *patch*) is either extinct or at its carrying capacity (i.e., cannot support any more individuals), and we otherwise ignore individual patch behaviour. We also assume that there is no spatial ordering of the patches, so there is an equal probability of migration to all other patches. If we denote the migration rate λ (which is the same for all patches) and the patch extinction probabability ϵ (also the same for all patches), we arrive at,

$$\frac{dP}{dt} = \lambda P(1-P) - \epsilon P \tag{2.11}$$

which also bears a strong resemblance to the logistic curve. From this equation, it follows that the equilibrium value of P is

$$\hat{P} = 1 - \frac{\epsilon}{\lambda} \tag{2.12}$$

The main ramification of this equation is that for a metapopulation to survive, then for a given extinction rate the migration rate must exceed a certain threshold. Hanski goes on to modify this model without the assumptions of homogenous migration and extinction rates [17].

Harrison [21] identifies several different kinds of metapopulations: 1) ”classic”, 2) mainland-island, 3) non-equilibrium, and 4) patchy (see Figure 2.5). ”Classic” metapopulations are those described by the standard metapopulation models such as Levins. The mainland-island metapopulation consists of a very large patch with guaranteed population persistence surrounded by a number of smaller patches on which populations are likely to die out. This type of metapopulation (also called ”source-sink”) describes a situation where individuals are frequently migrating away from a large population and will establish colonies that will eventually die out. Non-equilibrium metapopulations are made up of subpopulations that are so isolated from each other that inter-patch migration is unable to overcome the effect of extinction. Patchy metapopulations are such that the migration between patches is so high that effectively the metapopulation can be considered to be one population, with little chance of becoming extinct.

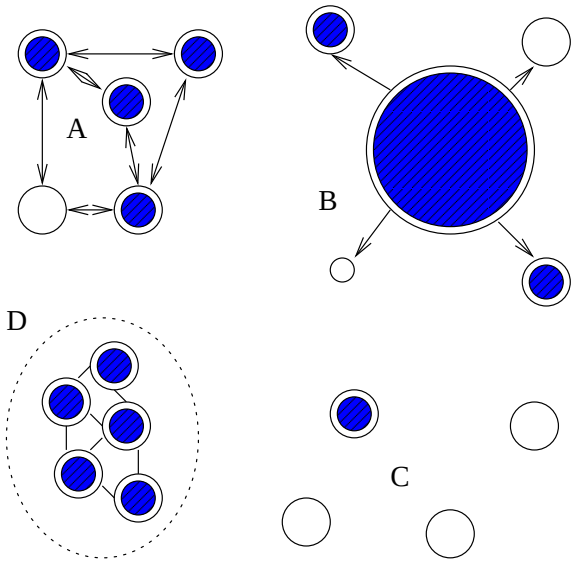


Figure 2.5: Harrison’s metapopulation types. Circles are patches; filled is occupied, unfilled is unoccupied. Arrows show routes of migration. A, Classic. B, Mainland-island. C, Non-equilibrium. D, Patchy

This section has just brushed the surface of metapopulation ecology; for a more thorough review, the reader is encouraged to consult either our previous paper [4] or more foundational works [19], [20], and [18].

The different metapopulation types are interesting because they resemble different topologies that have been implemented in PGAs. As the migration rate in a PGA changes, the topology can

change from resembling non-equilibrium metapopulations (when migration rate is very low), to classic metapopulations with a middle migration rate, to patchy metapopulations when the migration rate is very high.

Chapter 3

Mathematical analysis of information propagation on networks

3.1 Notation and Statements of the Main Problems

Throughout the following let G denote a network (a network is just an undirected graph) and let S denote a subset of nodes of G. Suppose every node in S is coloured red and the rest of the nodes are green. Suppose also that in a single time step any node from S can change the colour of any of its green neighbor into red with probability p and does not do it with probability $1-p$. Once a node is red it remains red. We are interested in the expected time it takes until all nodes in the network become red. We denote this waiting time random variable by T_S. We are then interested in computing $E(T_S)$.

3.2 Recursive Solution and some Simple Examples

An exact solution to the problem stated in section 3.1 can be obtained recursively using the following general methodology:

Theorem 3.2.1 *Suppose $\Omega = (\Omega, \mathcal{A}, P)$ is a probability space and $X_1,\ X_2, \ldots, X_n$ is a sequence of nonnegative integer-valued random variables on Ω (i. e. a nonnegative integer valued discrete stochastic process). Fix a subset $I \subseteq \mathbb{N}$ and let $\tau = \min\{n \,|\, X_n(\omega) \in I\}$. We are interested in computing $E(\tau)$.*

Let $\Omega_i = \{\omega \,|\, X_1(\omega) = i\}$ denote the conditional probability space with respect to X_1. We write $E_i(f)$ to denote the expectation of the random variable f with respect to the conditional probability space. Consider the shifted stochastic process $Y_1 = X_2$, $Y_2 = X_3$ ... $Y_n = X_{n+1}$ ... on Ω_i Let $\tau_i(\omega) = \min\{n \,|\, Y_n(\omega) \in I\}$. We then have:

$$E(\tau) = \sum_{i=0}^{\infty} P(X_1 = i) \cdot E_i(\tau_i) + 1.$$

Proof: This follows straight from the definitions:

$$E(\tau) = \sum_{j=1}^{\infty} P(\tau = j) \cdot j =$$

$$= \sum_{j=1}^{\infty} (\sum_{i=0}^{\infty} P(X_1 = i) \cdot P(\tau = j \mid X_1 = i)) \cdot ((j-1)+1) =$$

$$= \sum_{i=0}^{\infty} P(X_1 = i) (\sum_{j=1}^{\infty} \cdot P(\tau = j \mid X_1 = i) \cdot (j-1)+1)) =$$

$$= \sum_{i=0}^{\infty} P(X_1 = i) (\sum_{j=1}^{\infty} \cdot P(\tau = j \mid X_1 = i) \cdot (j-1)) + \sum_{i=0}^{\infty} P(X_1 = i) =$$

$$= \sum_{i=0}^{\infty} P(X_1 = i) \cdot E_i(\tau_i) + 1$$

□

Notice that the problem described in section 3.1 fits naturally into the framework of theorem 3.2.1. Indeed, the state space of our stochastic process (which is a Markov chain) consists of the set of all subsets of nodes of the graph G and the transition probability of passing from a set S to a set S' is the probability that the nodes in S' are coloured red and all the nodes in the complement of S' are green after a single time step provided that all the nodes in S are red and all the nodes in the complement of S are green. There are then finitely many states and they can all be enumerated by positive integers. The probability space consists of all sequences of states. The random variable X_i returns the number of the i^{th} state of the argument (i. e. the state which the system is in upon the completion of i time steps). Suppose the initial state of the system (at time 0) is $(G,\, S)$. For a given node $v \in v(G)$ denote by k_v the number of neighbors of v which are the members of the set S. Since a given node v may be coloured red independently by any of its neighbors which are in S with probability p, the probability that a given node $v \notin S$ stays green is $(1-p)^{k_v}$. Hence, the probability that v is coloured red is $1-(1-p)^{k_v}$. A node which is already in S stays red with probability 1 and so the only states which can occur after the first time step are the supersets $S' \supseteq S$. Since every node in S' is coloured red independently, the conditional probability

$$P(S'|S) = \begin{cases} (\prod_{v \in S'-S} (1-(1-p)^{k_v})) \cdot \prod_{v \notin S'} (1-p)^{k_v} \text{ if } S' \supseteq S \\ 0 \text{ otherwise.} \end{cases}$$

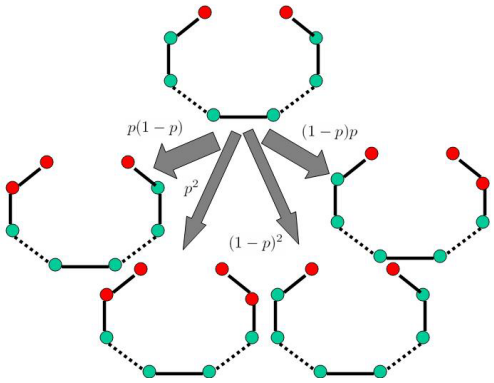
Applying theorem 3.2.1 to this situation we obtain the following recursive equation for the desired expectation:

$$E(T_S) = \sum_{S' \supset S} (\prod_{v \in S'-S} (1-(1-p)^{k_v})) \cdot (\prod_{v \notin S'} (1-p)^{k_v}) \cdot E(T_{S'}) + 1.$$

Unfortunately this equation is, in general, very complicated and computationally expensive. In some very simple cases we are able to solve this recursion though:

Example 3.2.1 *Consider for example the "line segment" graph where the set of nodes is* $V(L) = \{1, 2, \ldots, n\}$ *and the set of edges is* $E(L) = \{(i, i+1) \mid 1 \leq i \leq n-1\}$. *Suppose node* 1 *is coloured red and the rest of the nodes are green. Upon completion of the first step we obtain an identical line segment of length* n *with probability* $1-p$ *and, if the second node is coloured red, the situation is equivalent to the original problem with the line segment having length* $n-1$. *To reduce the complexity of notation, denote the desired expectation for a line segment of length* n *by* $E(n)$. *We then obtain the following recursive equation for a line segment of length* n*:* $E(n) = (1-p)E(n) + pE(n-1) + 1$. *If* $n = 1$ *then there is only one node which is already red and so* $E(1) = 0$. *The equations above then simplify to* $E(n) = E(n-1) + \frac{1}{p}$ *with* $E(1) = 0$ *and we readily see that* $E(n) = \frac{n-1}{p}$.

Example 3.2.2 *Now consider the "circle" graph: A circle on* n *nodes is obtained from the line on* n *nodes by connecting the nodes* 1 *and* n *by an edge (see example 3.2.1). Again we assume that node one is coloured red and the rest of the nodes are green. Notice that this problem is equivalent to the one where we are given a line segment of length* $n+1$ *and both ends (nodes* 1 *and* $n+1$*) are coloured red while the rest of the nodes are green. Then with probability* $(1-p)^2$ *nothing changes, with probability* $2p(1-p)$ *we are left with a line segment with both ends coloured of length* n *and with probability* p^2 *we are left with a line segment of length* $n-1$ *(see the picture below).*



According to our recursion formula we then obtain

$$E(n+1) = (1-p)^2 E(n+1) + 2p(1-p)E(n) + p^2 E(n-1) + 1.$$

Notice that $E(1) = E(0) = 0$. *The solution to this equation can be given exactly:*

$$E(n) = \frac{n}{2p} - (2-p)\left(1 - \left(\frac{p}{p-2}\right)^n\right)$$

or, asymptotically, $E[n] \approx \frac{n}{2p} + p - 2$.

In the next section we apply the recursive technique to obtain a solution for a hub network.

3.3 Hub Network

Definition 3.3.1 *A hub network is a graph consisting of a central node v which is connected to the rest of n nodes (there are totally $n+1$ nodes), and neither of these remaining n nodes are connected among each other. Formally the set of nodes is $V(G) = \{1, 2, \ldots, n+1\}$ and the set of edges is $E(G) = \{(1, i) \mid 2 \leq i \leq n+1\}$. We assume that initially the central node (node 1) is coloured red and the rest of the nodes are green. We also denote the desired expectation for the hub problem by $E(n)$ (the hub consists of $n+1$ nodes).*

The expected running time $E(n)$ for the hub problem satisfies the equation

$$E(n) = 1 + \sum_{k=0}^{n} \binom{n}{k} p^{n-k} q^k E(k)$$

which we rearrange to give the recurrence relation:

$$(1 - q^n)E(n) = 1 + \sum_{k=0}^{n-1} \binom{n}{k} p^{n-k} q^k E(k)$$

with $E(0) = 0$. The transmission probability is p and $q = 1 - p$. We claim that $E(n) = \Theta(\log(n+1))$.

We first prove the following interesting result:

Lemma 3.3.1 *If k is distributed according to the Binomial distribution $B(n, q)$, then the expected value of $\log(k+1)$ is $\Theta(\log(n+1))$. In particular,*

$$\log(n+1) - 1/q \leq \sum_{k=0}^{n} \binom{n}{k} p^{n-k} q^k \log(k+1) \leq \log(n+1) - \log\left(\frac{2}{1+q}\right)$$

(assuming natural logs — for other bases, the constant in the lower bound has to change accordingly).

Proof: To prove the upper bound, first note that, for $0 < q < 1$, and for all $n \geq 1$

$$nq + 1 \leq \frac{(n+1)(1+q)}{2}$$

Then, by concave property of logs,

$$\begin{aligned}
\sum_{k=0}^{n} \binom{n}{k} p^{n-k} q^k \log(k+1) &\leq \log\left(\sum_{k=0}^{n} \binom{n}{k} p^{n-k} q^k (k+1)\right) \\
&= \log(nq+1) \\
&\leq \log\left(\frac{(n+1)(1+q)}{2}\right) \\
&= \log(n+1) - \log\left(\frac{2}{1+q}\right)
\end{aligned}$$

Now note that the lower bound holds, if and only if:

$$\sum_{k=0}^{n} \binom{n}{k} p^{n-k} q^k \log\left(\frac{n+1}{k+1}\right) \leq 1/q$$

which we prove as follows:

$$\begin{aligned}
\sum_{k=0}^{n} \binom{n}{k} p^{n-k} q^k \log\left(\frac{n+1}{k+1}\right) &= \sum_{k=0}^{n} \binom{n}{k} p^{n-k} q^k \log\left(1 + \frac{n-k}{k+1}\right) \\
&\leq \sum_{k=0}^{n} \binom{n}{k} p^{n-k} q^k \left(\frac{n-k}{k+1}\right) \\
&= \sum_{k=0}^{n} \binom{n}{k} p^{n-k} q^k \left(\frac{n}{k+1}\right) - \sum_{k=0}^{n} \binom{n}{k} p^{n-k} q^k \left(\frac{k}{k+1}\right) \\
&\leq n \sum_{k=0}^{n} \binom{n}{k} p^{n-k} q^k \left(\frac{1}{k+1}\right) \\
&= \frac{n}{n+1} \sum_{k=0}^{n} \binom{n+1}{k+1} p^{n-k} q^k \\
&= \frac{n}{n+1} \sum_{k=1}^{n+1} \binom{n+1}{k} p^{n-k+1} q^{k-1} \\
&\leq \frac{1}{q} \cdot \frac{n}{n+1} \sum_{k=0}^{n+1} \binom{n+1}{k} p^{n+1-k} q^k \\
&\leq \frac{1}{q}
\end{aligned}$$

(where we used the fact that $\log(1+x) \leq x$ for all $x \geq 0$). □

Theorem 3.3.2 $E(n) = \Theta(\log(n+1))$. *In particular,*

$$q \log(n+1) \leq E(n) \leq \left(\frac{1}{\log \frac{2}{1+q}}\right) \log(n+1)$$

Proof: We prove, by induction, that $E(n) \leq A \log(n+1)$, where

$$A = \frac{1}{\log \frac{2}{1+q}}$$

The case $n = 0$ is easy, since $E(0) = 0 = A \log 1$. Now suppose $n \geq 1$ and that the hypothesis is true for all $0 \leq k < n$. Then

$$\begin{aligned}
(1 - q^n)E(n) &\leq 1 + \sum_{k=0}^{n-1} \binom{n}{k} p^{n-k} q^k A \log(k+1) \\
&= 1 - Aq^n \log(n+1) + A \sum_{k=0}^{n} \binom{n}{k} p^{n-k} q^k \log(k+1) \\
&\leq 1 - Aq^n \log(n+1) + A \log(n+1) - A \log\left(\frac{2}{1+q}\right) \\
&= A(1 - q^n) \log(n+1)
\end{aligned}$$

and the result follows.

Secondly, we show by induction that $E(n) \geq q \log(n+1)$. The case $n = 0$ is again easy. Now suppose the hypothesis is true for all $0 \leq k < n$. Then

$$\begin{aligned}
(1 - q^n)E(n) &\geq 1 + q \sum_{k=0}^{n-1} p^{n-k} q^k \log(k+1) \\
&= 1 - q^{n+1} \log(n+1) + q \sum_{k=0}^{n} p^{n-k} q^k \log(k+1) \\
&\geq 1 - q^{n+1} \log(n+1) + q \left(\log(n+1) - \frac{1}{q}\right) \\
&= q(1 - q^n) \log(n+1)
\end{aligned}$$

and the result follows. □

3.4 Complete, Bipartite and Multipartite Graphs

Definition 3.4.1 *Suppose we are given a set V on kn elements which is partitioned into k subsets consisting of n elements each. Denote this partition by $\mathcal{P} = \{K_1, K_2, \ldots, K_k\}$. A graph $G = (V, E)$ is called a k-partite graph on kn vertices if whenever $(u, v) \in E$ with $u \in K_i$ and $v \in K_j$ we have $\forall\, x \in K_i$ and $y \in K_j$ $(x, y) \in E$. A multipartite graph is called complete if $\forall\, i \neq j$ we have $(x, y) \in E$ whenever $x \in K_i$ and $y \in K_j$. If every vertex of K_i is connected to every vertex of K_j we say that K_i is a neighbor of K_j (according to our definition if at least one vertex of K_i is connected to at least some vertex of K_j then all of the vertices of K_i are connected to all of the vertices of K_j).*

Notice that according to definition 3.4.1, every multipartite graph is completely determined by specifying a partition $\mathcal{P}$ and specifying a set of neighbors for every element of the partition. it makes sense, therefore, to introduce the following definition:

Definition 3.4.2 *Let G denote a multipartite graph with partition $\mathcal{P} = \{P_1, P_2, \ldots, P_k\}$. A prototype of G is the graph G' whose set of nodes is $\mathcal{P}$ and $(P_i, P_j) \in E(G')$ if and only if P_i is a member of P_j in the sense of definition 3.4.1.*

Throughout the following n denotes the number of nodes inside of each element of the partition. Below are some examples:

Example 3.4.1 *If $\mathcal{P} = \{P\}$ is a singleton set and P is a member of itself then we obtain a complete graph on n nodes (every node is connected to every other node). In other words, a complete graph on n nodes is just a multipartite graph whose prototype graph consists of a single node and a loop edge.*

Example 3.4.2 *Suppose $\mathcal{P} = \{P, Q_1, Q_2, \ldots, Q_k\}$ and suppose P is connected to every Q_i but Q_i's are not connected to each other. We call such a graph a multipartite hub. In other words, a multipartite hub is just a multipartite graph whose prototype is a hub.*

Example 3.4.3 *A complete multi-graph is a multipartite graph whose prototype graph is a complete graph (not to be confused with the notion of a complete graph).*

In this section we investigate the asymptotic propagation times as $n \to \infty$ for multipartite graphs where the network topology as well as the number of elements in the partition $\mathcal{P}$ stay fixed (i. e. the prototype graph stays fixed). We always assume that one of the nodes inside of one of the elements of the partition is coloured red.

The results of the current section are based on the following facts known as Chernoff bounds (see, for instance, section 4.1 of [30]):

Theorem 3.4.1 (Multiplicative Chernoff Bound) *Let $X_1, X_2, \ldots, X_n$ be independent Poisson trials with $Pr(X_i = 1) = p_i$ and $Pr(X_i = 0) = 1 - p_i$. Then if $X = \sum_{i=1}^{n} X_i$ and if μ is $E(X)$, for any $\delta \in (0, 1]$ we have*

$$Pr(X < (1 - \delta)\mu) < e^{\frac{-\mu\delta^2}{2}}.$$

The following fact has been obtained in [10] and in [22].

Theorem 3.4.2 (Additive Chernoff Bound) *Let $X_1, X_2, \ldots, X_n$ be independent Poisson trials with $Pr(X_i = 1) = p_i$ and $Pr(X_i = 0) = 1 - p_i$. Then if $X = \sum_{i=1}^{n} X_i$ and if μ is $E(X)$, for any λ with $0 \leq \lambda < n - \mu$ we have*

$$Pr(X \leq \mu - \lambda) \leq \exp(nH_{1-p}(1 - p + \frac{\lambda}{n}))$$

where $p = \frac{\mu}{n}$ and $H_p(x) = x \ln(\frac{p}{x}) + (1 - x) \ln(\frac{1-p}{1-x})$ is the relative entropy of x with respect to p.

By letting $p_i = p \ \forall\, i$ in the statement of theorems 3.4.1 and 3.4.2 and observing that the sum of independent and identically distributed Poisson trials with probability p each is distributed binomially (with probability of success p, probability of failure q and mean np), we immediately deduce the following:

Corollary 3.4.3 *let X denote a binomially distributed random variable with probability of success p and probability of failure $q = 1 - p$. Then*

$$Pr(X < (1-\delta)pn) < e^{-\frac{np\delta^2}{2}}$$

and

$$Pr(X \leq np - \lambda) \leq \exp(nH_{1-p}(1 - p + \frac{\lambda}{n})).$$

We now proceed applying corollary 3.4.3 to the case of complete and multipartite graphs. The main results then are the following:

Proposition 3.4.4 *Given a k-partite graph on kn vertices with partition $\mathcal{P} = \{P_1, P_2, \ldots, P_k\}$, suppose $Q_1, Q_2 \ldots Q_j$ are the neighbors of a member P of the partition. Suppose exactly one node in P is coloured red. Then, upon completion of a single time step, Pr(that number of nodes inside of every Q_i which are not coloured red is smaller than $n\frac{p}{2}$) $= O(e^{-n})$. Therefore Pr(that number of nodes inside of every Q_i which are coloured red is at least $n\frac{p}{2}$) $\geq 1 - O(e^{-O(n)})$. Moreover, Pr(that the number of green nodes remaining inside of every Q_i is at least $n\frac{q}{2}$) $\geq 1 - O(e^{-O(n)})$.*

Proof: The first assertions follow immediately from corollary 3.4.3 with $\delta = \frac{1}{2}$ when we observe that the number of red colour nodes inside of every Q_j after a single time step is distributed binomially with probability p of success while the number of green nodes is distributed binomially with probability of success q. □

Proposition 3.4.5 *Given a k-partite graph on kn vertices with partition $\mathcal{P} = \{P_1, P_2, \ldots, P_k\}$, suppose $Q_1, Q_2 \ldots Q_j$ are the neighbors of a member P of the partition. Suppose at least $n\frac{p}{2}$ nodes in P is coloured red. Then all the vertices in every Q_i are coloured red in a single time step with probability $\geq 1 - O(e^{-O(n)})$.*

Proof: Any given node of Q_i is not coloured red with probability at most $(1-p)^{n\frac{p}{2}} = q^{an}$ with $q = 1 - p$ and $a = \frac{p}{2}$ and so every node of Q_i is coloured red with probability at least $1 - q^{an}$. Thus, the number of nodes coloured in every Q_i is distributed binomially with probability of success at least $1 - q^{an}$. Choose N large enough that $\forall\, n > N$ we have $q^{an}n < 0.4$ and let $\lambda = 0.5$. Applying corollary 3.4.3 to this distribution and observing that $n(1 - q^{an} - \lambda = n - nq^{an} - 0.5 > n - 0.9$ for $n > N$ gives us Pr(less than $n - 0.9$ nodes are coloured red after a single time step) $< \exp nH_{q^{an}}(q^{an} + \frac{\lambda}{n})$. Now lets examine the behavior of the functions $H_{q^{an}}(q^{an} + \frac{\lambda}{n})$ as $n \to \infty$: By definition we have

$$H_{q^{an}}(q^{an} + \frac{\lambda}{n}) = (q^{an} + \frac{\lambda}{n}) \ln(\frac{q^{an}}{q^{an} + \frac{\lambda}{n}}) + (1 - q^{an}) \ln(\frac{1 - q^{an}}{1 - q^{an} + \frac{\lambda}{n}}).$$

Notice that $\frac{1-q^{an}}{1-q^{an}+\frac{\lambda}{n}} \to 1$ as $n \to \infty$ so that $\ln(\frac{1-q^{an}}{1-q^{an}+\frac{\lambda}{n}}) \to 0$ as $n \to \infty$ and we get $(1 - q^{an}) \ln(\frac{1-q^{an}}{1-q^{an}+\frac{\lambda}{n}}) \to 0$ as $n \to \infty$. We then deduce that

$$\lim_{n\to\infty} H_{q^{an}}(q^{an} + \frac{\lambda}{n}) = \lim_{n\to\infty} (q^{an} + \frac{\lambda}{n}) \ln(\frac{q^{an}}{q^{an} + \frac{\lambda}{n}}) =$$

$$= \lim_{n\to\infty}(q^{an}+\frac{\lambda}{n})(an\ln q - \ln(q^{an}+\frac{\lambda}{n})) = \lambda a \ln q$$

which is just a fixed negative number. Let $\kappa = \frac{\lambda a \ln q}{2}$ and notice that for all large enough n we finally have Pr(less than $n-0.9$ nodes are coloured red after a single time step) $< \exp(\kappa n)$ which immediately leads to the desired inequality Pr(all nodes are coloured red after a single time step) $< 1 - \exp(\kappa n)$. □

Example 3.4.4 *Suppose we are given a complete graph G on n vertices. Recall from example 3.4.1 that such a graph can be regarded as a multipartite graph with partition $\mathcal{P} = \{P\}$ being a singleton set and P being its own neighbor. Suppose one of the nodes in P is coloured red. Then, according to proposition 3.4.4, after the first time step $n\frac{p}{2}$ nodes are infected with probability $\geq 1 - O(e^{-O(n)})$. Given that this has happened, after the second time step all of the nodes in P are coloured with probability $\geq 1 - O(e^{-O(n)})$. Therefore the probability that everyone is coloured after two time steps is at least $(1-O(e^{-O(n)}))(1-O(e^{-O(n)})) = 1-O(e^{-O(n)})$.*

Theorem 3.4.6 *Let G denote a multipartite graph without loops (meaning no element of the partition is connected to itself). Denote by G' the prototype graph of G and suppose one of the nodes inside a member P of the partition is coloured red. Suppose the eccentricity of P is at least 3. Then, if we let $n \to \infty$ (the size of every element of the partition) while keeping the prototype graph G' fixed, the time it takes to colour all nodes red with probability $1 - O(e^{-O(n)})$ approaches the eccentricity of the node P in the prototype graph G'. If the eccentricity of the node P is either 2 or 3 then it takes exactly 2 time steps if and only if every neighbor of P is a neighbor of another neighbor of P. Otherwise it takes 3 time steps.*

Proof: In order to prove theorem 3.4.6 we establish the following claim by induction:

Claim: Denote by $\mathcal{Q}(l)$ the set of all nodes of G' which are at most l steps away from P (meaning that the shortest path from P to every $Q \in \mathcal{Q}(l)$ in G' consists of at most l edges) then as $n \to \infty$ the following holds:

1. If $l = 1$ or $l = 2$ it takes either 2 or 3 time steps to colour every node inside of every $Q \in \mathcal{Q}(l)$ red with probability $1 - O(e^{-O(n)})$. In fact, it takes 2 steps if and only if every $Q \in \mathcal{Q}(1)$ has a neighbor in $\mathcal{Q}(1)$ Moreover, the probability that all nodes inside of every $Q \in \mathcal{Q}(l)$ are coloured red after a single time step is exponentially small in n. Finally, every node inside of Q which is exactly 2 edges away from P is coloured after 2 time steps and every node inside of P is coloured after 2 time steps.

2. If $l \geq 3$ then it takes l steps to colour every node inside of every member of $\mathcal{Q}$ red with probability $1 - O(e^{-O(n)})$.

If $l = 1$ or 2, let R be any immediate neighbor of P. According to proposition 3.4.4, after the first time step at least $n\frac{p}{2}$ nodes are coloured red and at least $n\frac{q}{2}$ nodes remain green with probability $1 - O(e^{-O(n)})$. Then, upon the completion of the second time step, according to proposition 3.4.5, all of the nodes inside of every neighbor of R (including P itself) are coloured red with probability $1 - O(e^{-O(n)})$. The only nodes which possibly remain green with high probability are these which are inside of R itself. In fact, unless R is connected to another immediate neighbor of P, since at least $n\frac{q}{2}$ nodes remain green with probability $1 - O(e^{-O(n)})$,

a routine application of Chernoff bound (see theorem 3.4.1) shows that still a positive fraction of nodes remains green inside of R after the second time step. It then requires 3 time steps to colour everyone red. If, on the other hand, a given node R has a neighbor T which is also an immediate neighbor of P, then after the first time step, at least $n\frac{p}{2}$ of the nodes in both, T and R have been coloured red with probability $1 - O(e^{-O(n)})$. But then, after the second time step they both fill each other and all of their neighbors with red colour with probability $1 - O(e^{-O(n)})$. In this case it takes only 2 time steps.

Now suppose $l \geq 3$ and the statement holds for $l - 1$. Let $Q \in \mathcal{Q}(l)$ be exactly l edges away. Let $\mathcal{S}(Q)$ denote the set of all neighbors of Q and $\mathcal{N}(Q)$ the set of all neighbors of Q which are exactly $l - 1$ edges away from P. By inductive hypothesis it takes exactly $l - 1$ time steps to colour every node of every member of $\mathcal{N}(Q)$ red (this is true even if $l - 1 = 2$). According to proposition 3.4.5 upon completion of the next (l^{th} step) every node in Q will be coloured red with probability $1 - O(e^{-O(n)})$. Even if $l - 1 = 2$ everyone of the nodes in $\mathcal{Q}(2)$ is guarantied to be coloured red upon the completion of the 3^{rd} time step. The desired conclusion then follows by the principle of induction. □

As a special case of theorem 3.4.6 we immediately conclude that a multipartite hub (see example 3.4.2) with any fixed member of partition P containing a single red node takes 3 time steps to colour everyone red with probability $1 - O(e^{-O(n)})$. Notice that this includes the bipartite graph as well. Likewise, another special case of theorem 3.4.6 shows that for any complete multi-graph (see example 3.4.3) it takes exactly 2 time steps before everyone becomes red with probability $1 - O(e^{-O(n)})$.

3.5 Simple upper bounds

Consider a tree of depth h and branching factor b. An upper bound for the propagation time on such a tree (with the information initially at the root) can be calculated by viewing it as a series of hubs. The first hub is to the immediate neighbours of the root. Propagation to all of these takes$O(\log b)$. We can then view the propagation to the next level of the tree as a hub with b^2 nodes. This takes $O(\log b^2)$. Continuing, we get an upper bound of

$$\log b + 2\log b + 3\log b + \cdots + h\log b = O(h^2 \log b)$$

For example, a balanced binary tree has $b = 2$ and $h = \log n$, and so would have a propagation time bounded above by $O((\log n)^2)$.

We can use this approach to get a quick upper bound on general graphs, by considering the information flow along a spanning tree rooted at the initial node. For example, a random graph of fixed-degree has a diameter $O(\log n)$ and so this gives a bound on the propagation time the same as for the binary tree. A square grid, on the other hand, has diameter $\sqrt{(n)}$ which gives abound of $O(n)$ on the propagation time. However, these upper bounds are not necessarily very tight. A lower bound is, of course h, the eccentricity of the initial node. For the grid this is $\sqrt{n}$ and so we see that it cannot be as efficient as a random graph.

We are currently working on a tighter bound, in collaboration with Dr. Leslie Goldberg (University of Warwick). This will be the subject of a future report.

3.6 A Note on Random Walks on Graphs

In this section we mention a different problem which has been well-studied in the literature. Here we are given a graph $G = (V, E)$ where $V = \{v, x, y, \dots\}$ is a finite set of nodes and $E = (e_1, e_2, \dots)$ is the set of edges. We assume that the graph is not directed and has no multiple edges and no self-loops. In a weighted graph G each edge (v, x) also has a weight $w_{(v,x)} = w_{(x,v)} > 0$, and we allow a weighted graph to have self-loops.

Given a weighted graph, there is a natural definition of a Markov chain on the nodes. Define a discrete time random walk on a weighted graph to be the Markov chain with transition matrix

$$p_{v \to x} = \begin{cases} \frac{w_{vx}}{w_v} \text{ if } x \text{ is a neighbor of } v \\ 0 \text{ otherwise} \end{cases}$$

where $w_v = \sum_{x \text{ is a neighbor of } v} w_{vx}$. Let w be the total edge weight so that each edge is counted *twice*, i. e., twice in each direction. The fundamental fact is that this chain is automatically reversible with stationary distribution $\pi_v = \frac{w_v}{w}$. From the general Markov chain theory we know various identities for the mean hitting times: For example, if one starts running the chain starting at the state $x \in A \subseteq V$ then the expectation of the mean hitting time $T_A^+ = \min\{t \geq 1 \mid X_t \in A\}$ for returning to A provided that the chain originated at A and the states from A are chosen with respect to the stationary distribution, π_A, conditioned on A ($\pi_A(x) = \frac{\pi(x)}{\pi(A)}$) is given by the Kac's formula: $E_{\pi_A}(T_A^+) = \frac{1}{\pi_A}$. In particular, if the chain is started at the state $x \in V$ then $E(T_x^+) = \frac{1}{\pi(x)}$.

In case every weight $w_{xy} = 1$ and there are no loops, we have

$$p_{vx} = \begin{cases} \frac{1}{d_v} \text{ if } (v, x) \text{ is an edge} \\ 0 \text{ otherwise} \end{cases}$$

and so $\pi_v = \frac{d_v}{2|E|}$.

Now suppose a piece of information is traveling around a network and passes from one node to another with probability proportional to $\frac{1}{d_v}$ where d_v is the degree of the node where the information is currently located. The results above apply immediately to conclude that the expected return time to the original node v is $\frac{2|E|}{d_v}$. Here is another example:

Lets say that there are 2 gossips starting to spread around from the same node, say v independently with probabilities proportional to the corresponding degrees of the nodes they pass through. Suppose we want to know how often on average they meet. To solve this problem we apply Kac's formula to the product of the weighted graph G with itself:

Definition 3.6.1 *Given weighted graphs $G = (V_G, E_G)$ and $H = (V_H, E_H)$, build another weighted graph $G \times H = (V_G \times V_H, E_G \times E_H)$ and every edge $((g_1, h_1)(g_2, h_2)) \in E_G \times E_H$ has the weight $w_{g_1h_1} \cdot w_{g_2h_2}$. We write G^2 to mean $G \times G$.*

Let $A = \{(x,\, x) \,|\, x \in V_G\}$ denote the diagonal subset of nodes. Our problem then boils down to computing the mean return time to A starting at A with the stationary distribution. We have then $\pi(A) = \sum_{x \in V_G} \frac{d_x^2}{4|E|^2}$ so that, according to Kac's formula, the desired expectation

$$E_{\pi_A}(T_A^+) = \frac{4|E|^2}{\sum_{x \in V_G} d_x^2}.$$

Chapter 4

Empirical studies of network propagation time

We created an information diffusion model and simulated diffusion on different network topologies. There are two distinct studies. The first continues the work of the previous chapter in looking at the dependence of propagation time on the number of nodes in the network. Here we consider *scale-free* networks. In the second set of studies we change our focus to the dependence on the transmission probability p. We so far have no detailed theory of this, so empirical results for a variety of network topologies are given. We also look at some alternative models of propagation (e.g. when the information goes to all neighbours in a single transmission).

4.1 Dependence on number of nodes

The effect of the size of a scale-free network on transmission time was studied for graphs up to 350 nodes. These are constructed using the *preferential attachment* method. This is a recursive method, which begins with two connected nodes. Then, given a network with n nodes add the $n+1$st node (together with 2 edges sticking out of it) to the network by attaching the first edge to an existing node with probability proportional to that node's degree. Now once the first edge is attached to an existing node, say v, generate a new probability distribution on the set of all existing nodes minus v with the probability of attachment proportional to the node's degree again, and attach the remaining edge to one of these nodes with respect to this probability distribution.

For each size of graph, we ran experiments 40 times to estimate the propagation time. The initial information started at the node of maximal degree (although the results don't seem to depend on where it starts). The results for propagation probabilities 0.2, 0.5 and 0.8 are shown in figure 4.1. We have conjectured that the propagation time for this kind of network should be logarithmic in the size of the network (as is the diameter of scale-free networks). The empirical evidence supports this claim.

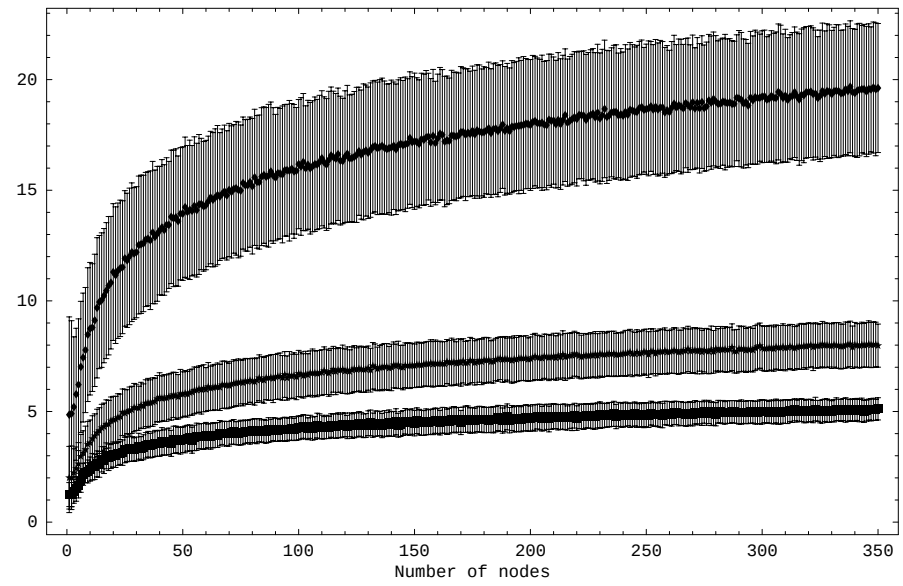


Figure 4.1: Propagation time for scale-free networks of different sizes. The three sets of results are for $p = 0.2$ (slowest), $p = 0.5$ and $p = 0.8$ (quickest). Averages over 40 runs are given, together with error bars of one standard-deviation.

4.2 Dependence on transmission probability

The model works by seeding vertices in the network with a unique piece of information. Each time step, the information has a probability (i.e., migration rate) to spread to one or more of its neighbours. The simulation is run until all nodes have received the information. All diffusion experiments were run on networks with 100 nodes.

It is expected that, especially for lower migration rates, the topologies with the larger characteristic path lengths will take longer for the information to diffuse, since the path that information must travel is longer.

4.2.1 Diffusion from Single Node

The first experiment observed how long it took for information to diffuse from a single node to the rest of the network. Each run, a random node was selected as the starting vertex. Every time step, each node that had already seen the information propagated it onwards, with a certain probability based on the migration rate, to a random one of its neighbours. Figure 4.2 shows the results for the different topologies.

The results show that the topologies with the shorter characteristic path lengths did in fact take in general much less time for the information to diffuse[1]. The exception is the scale-free

[1]For example, a small-world network has a much shorter characteristic path length than a grid network for the same number of nodes (calculations not shown).

graph, which took much longer than the other topologies. The reason for this is that with this experiment, nodes that have the information and are due to propagate it onwards only propagate it to a single neighbour. Scale-free graphs tend to have nodes in the ”center” with many connections, but connections to nodes which themselves have few connections. What this means is that the information tends to reach the central nodes very quickly, but it takes a long time for information in the central nodes to reach all of the outlying nodes. We would expect the situation to be reversed when information is propagated to all neighbours, which is is the subject of the next experiment.

4.2.2 Diffusion from Single Node to All Neighbours

In this experiment, a single node is again chosen to begin the diffusion, but if it is due to propagate, instead of choosing a single neighbour it propagates the information to all neighbours (Figure 4.3). We expected the scale-free network in this experiment to be one of the fastest, since once the information reaches the central nodes, it should very quickly reach the rest of the nodes.

As expected, the scale-free topology in this experiment, instead of being the slowest, was one of the fastest, due to the effect mentioned above. Also, notice that the ring topology was almost exactly twice as fast in this experiment as in the last; this makes sense since the information now propagates in both directions until it meets on the far side of the ring.

4.2.3 Diffusion from All Nodes

Next, we ran an experiment that starts every node with a unique piece of information and runs until all information has been propagated to all nodes. The results were expected to be similar to the single start results, except all runs would obviously take longer. They also should be a more ”general” test of the networks since in the single start experiment, the starting node might be in a special location that influences the result in that particular run (although since all experiments were run multiple times, this should not be that big of an issue). Figure 4.4 shows the results of this experiment.

As expected, the results were very similar to the first experiment, with a longer run time. The scale-free graph was significantly slower.

4.2.4 Diffusion from All Nodes to All Neighbours

For completeness sake, the last experiment (see Figure 4.5) measured the time it took for information from all nodes to diffuse to all nodes, diffusing to all neighbours each step.

The results from this experiment follow logically as a combination of the previous two. Similarly to the second experiment, the scale-free network performed much better than in the previous experiment.

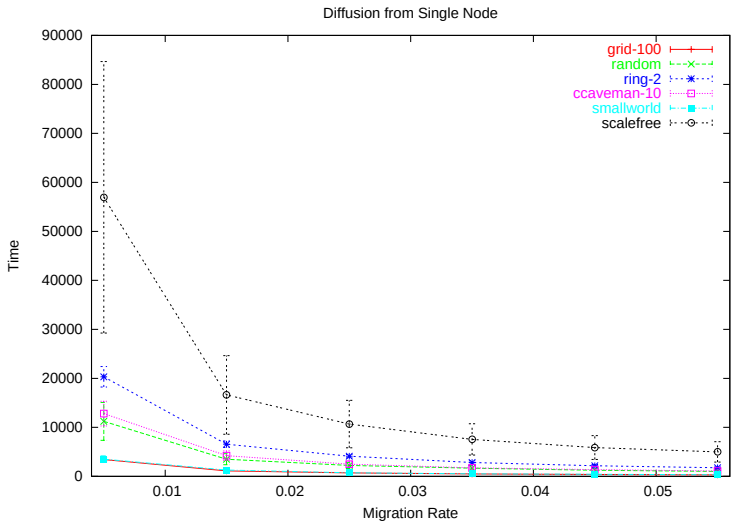


Figure 4.2: Diffusion from Single Node.

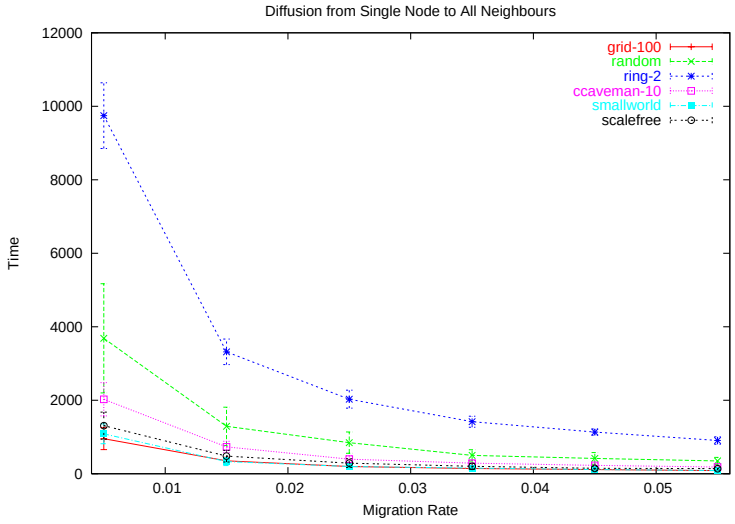


Figure 4.3: Diffusion from Single Node to All Neighbours.

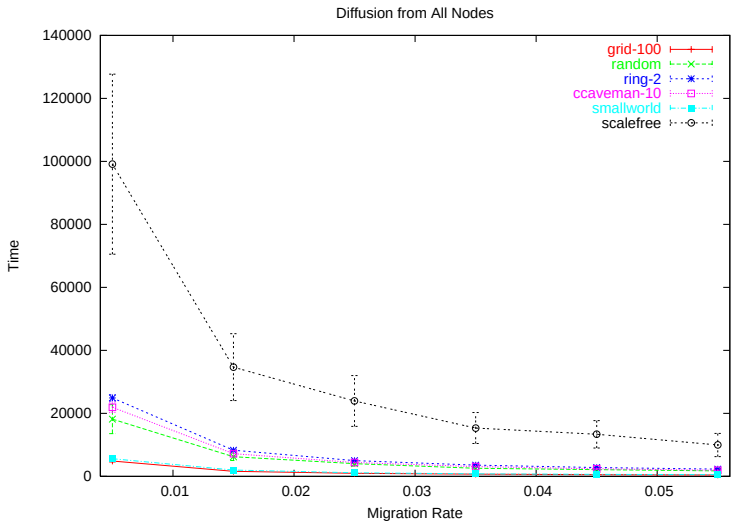


Figure 4.4: Diffusion from All Nodes.

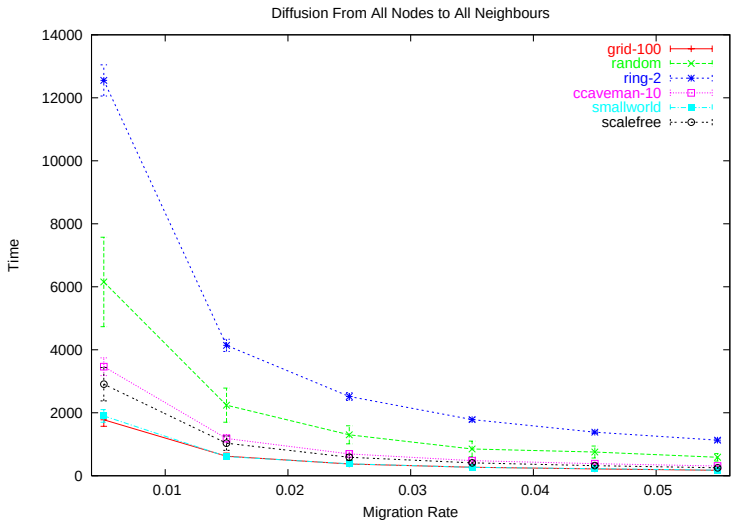


Figure 4.5: Diffusion from All Nodes to All Neighbours.

4.3 Relevance

The main ramifications of these results to the DBE project is that depending on the underlying network topology, the time it takes for information to diffuse spans a wide range. Small-world networks seem efficient empirically. We are currently working on verifying this claim analytically. The scale-free networks appear to scale well with increasing network size, but for smaller networks, they are slow with respect to the propagation probability. Grid networks, although they look good in the experiments, would probably not scale so well (we conjecture the propagation time for grids grows as $\sqrt{n}$). There is further work to be done on clearing up these cases. One key factor in applying the results to the DBE will be to get an estimate of the size of the networks concerned and their potential growth. Schemes which scale badly might still be the method of choice if the number of nodes is going to remain limited.

Chapter 5

A Parallel Genetic Algorithm for the DBE

This chapter presents a new model that is designed to be used for simulation of some aspects of the proposed DBE framework. Specifically, our model is a PGA meant to play the role of the search agent in finding software for SMEs. The focus of the model is the effect of changing the topology of the patches on information flow and eventual performance of the PGA, and thus the focus is on the connection structure of the patches. While it does not simulate all aspects of the proposed DBE framework, it is hoped that it will elucidate issues that might arise from particular topologies to aid in the eventual creation of the real DBE framework.

The first section will describe a modified version of the SCP that we use as the objective function, the second and third will discuss the specifics of the GA and PGA we use, and the last will describe the six network topologies we chose to use to connect the patches.

5.1 DBE Subset Covering Problem

As described above, the Set Covering Problem (SCP) attempts to find a subset of columns in a binary matrix whose union contain at least one 1 in all rows, as well as minimising the total cost (since each column has an associate cost). We propose a slight modification to the unicost SCP that introduces a ”target” bit string, where each bit position corresponds to a row in the matrix. The subset of columns only needs to cover those rows that have a corresponding ’1’ in the target string (see Figure 5.1; note that the matrix is transposed). The actual complexity and difficulty of the problem is now actually dependent on the number of 1’s in the bit-string, since a larger number of 1’s represents more rows that need to be covered and thus a harder problem; apart from that the underlying problem dynamics should be very similar.

Since our problem is unicost, the problem becomes first determining if there actually exists a set cover, and then minimising the *size* of that set (instead of the cumulative cost). An ideal solution to the DBE problem would be a set containing exactly one column that contained 1s in just the rows with a corresponding 1 in the target string and nothing else, where a degenerate

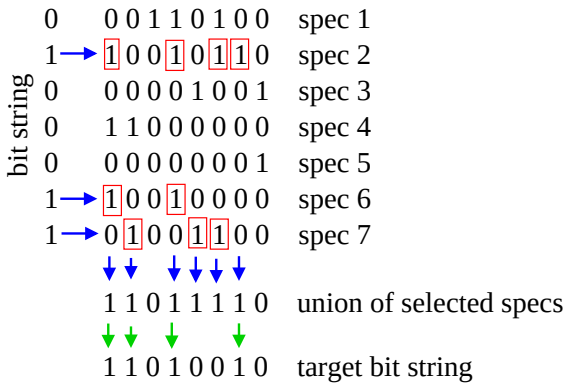


Figure 5.1: The DBE Subset Covering Problem. A bit string determines which specifications (rows) are in the subset. The union of all selected specifications is compared against the target bit string. If all positions in the target with a 1 have a corresponding 1 in the union string, the subset is a cover. (Note that this matrix is transposed from the description in the text)

solution might just include all columns in the set.

The rationale behind this reformulated SCP is to simulate the problem a search agent in the DBE framework will face when trying to match software specifications with a software requirement. Each column in the matrix represents a different software specification, and each row in the matrix represents a particular software capability; for instance, a particular piece of software may provide multiple capabilities - each of its capabilities is represented as a 1. The list of specifications is of configurable size, is generated randomly, and is the same for all patches. The target bit string represents an SME's software requirements, where each 1 represents that capability (i.e., row) being required.

Although this formulation of the SCP is unicost, the future business model of the DBE framework would probably associate some sort of individual cost with software packages. This cost might either be set by the software producers themselves; set as some sort of automatic market cost; or perhaps as a function of how many capabilities it provides. However, for the scope of this project each column was given the same cost.

5.2 Genetic Algorithm

Although there are many different methods for solving the SCP [7], we chose a GA due to its simplicity in implementation and ability to be parallelised. Additionally, the motivation behind the DBE project is to find nature-inspired methods and we felt that this was a good fit.

Genetic Representation

Following [7] and [37], we also made the obvious choice of using a bit string encoding for the genetic representation, since in this case the genome maps directly to the individual itself and does not require an expression process. The main drawback to this representation is that the length of the genome grows directly with the number of software specfications (columns); for a large number of possible specs this could quickly become intractable.

Fitness

For fitness evaluation, the genome was taken and for each 1 the corresponding software specification was added to a set. The union of all specifications in the set yielded a capability string, which was compared to the target bit string. Fitness is a sum of a) the number of specifications in the set (*size*), b) the number of 1s in the target string that are not in the capability string (*missCaps*), c) the number of 1s in the capability string that are not in the target string (*extraCaps*), and d) a set penalty if there are any 1s in the target string not in the capability string (*isCover*). Each of these has an associated coefficient. These are parameters of the GA (see Section 7.1.1). In other words,

$$F = \alpha S + \beta C_E + \delta C_M + \gamma I \tag{5.1}$$

where S is the size, C_E is the number of extra capabilities, C_M is the number of missed capabilities, and $I \in \{0, 1\}$. Since infeasible solutions (i.e., ones that do not meet all the capabilities) are allowed, this constitutes a penalty approach.

Selection/Crossover

Since finding the best assignment of penalty coefficients is a relatively difficult task, and the resulting fitness values can potentially be very diverse, we decided to use binary tournament selection since it relies only on relative fitness.

With two parents selected, we used uniform crossover to create one new offspring, with an equal weighting between the two parents. In addition to binary tournament selection, we also tested the GA with a different form of selection in which the best two individuals in the population were selected and the resulting offspring replaced the worst individual in the population (which we called ”elite” crossover). This tended to reduce diversity too quickly and thus we primarily used the former method (see Section 7.1.5 for our experimental results of the different crossover types).

Mutation

We tried two different forms of mutation: ”flip” mutation which for each individual selected to mutate (based on the mutation rate) we flipped each of its bit based on another probability (the

flip rate), and ”swap” mutation in which for each selected mutant we juxtaposed one or more (usually one) pair of bits.

Flip mutation creates much more diversity for the same mutation rate, obviously, since in swap mutation at most two bits will change (if flipping one pair).

5.3 Parallel Genetic Algorithm

Since the point of this project was to investigate the effects of different network topologies, we used a multiple-population model for our PGA. Each patch functions as a normal GA and has no knowledge of the other patches. The PGA contains a graph that maintains a list of neighbours for each patch. Each time step the PGA decides which individuals in which patches should migrate to which of its neighbours.

The main difference in our model with typical PGAs is the ability for different patches to have different target bit strings. This actually is a substantial difference that has far-reaching effects on the dynamics of the entire system, and in fact places it more in the realm of a metapopulation than a typical PGA.

Individual Selection

The PGA can either select the best individual in a population or a random individual. Selecting a random individual is more like nature but has potential drawbacks in terms of fitness, since it is often likely to introduce suboptimal individuals into the neighbouring populations. This would potentially increase diversity, however, so this is a trade-off.

Migration

We implemented two different migration types: copy and move. In move migration, the individual is actually moved to the destination patch and removed from its source population. This results in fluctuations in the patch population sizes, which for some network topologies can present problems. In copy migration, the individual is copied into the destination patch where it overwrites an existing individual.

When individuals are copy migrated, they can either be copied to one neighbour or to all neighbours.

5.4 Network Topologies

We chose six different possible network topologies for the PGA, which were designed to give a range of path lengths as well as realistic feasibilities. Below, we discuss the algorithms used to

generate these networks. The main constraint placed upon the generation algorithms is that since we are modelling migration, the graph must be connected.

Random

A random graph is simply a graph whose edges were generated randomly [40] (see Figure 5.2). This topology was chosen as a bounding case for the DBE if no design was followed in connecting nodes.

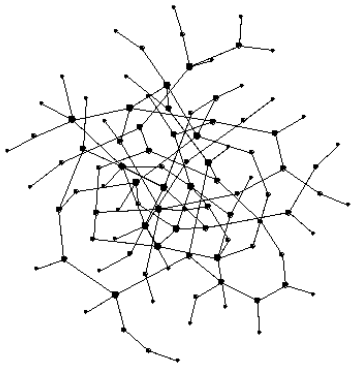


Figure 5.2: Random graph.

Since the graph needs to be connected, there must be at least $\frac{n(n-1)}{2}$ edges. Because each vertex will have at least one edge attached to it, we simply iterate through the vertices, creating a new edge from the current vertex (if it is not already connected) to a random distinct vertex. After that, if the graph is not connected we add edges between random vertices until it is connected. Potential inefficiencies aside, this method produces random graphs without a bias towards particular vertices (e.g., the initial nodes in a growth model).

Connected Caveman

The ”Connected Caveman” graph is a graph model proposed by Watts that presents a bounding case for social networks [40]. In it, there are multiple ”caves”, or groups of vertices, in which every vertex is connected to all the rest. These caves are linked together by single edges into a global ring (see Figure 5.3). This type of graph, with a very high clustering coefficient, is interesting because it creates a hierarchy of sorts in the topology in which information is likely to be very quickly passed around on a local scale but will take much longer to diffuse to the rest of the network; this bears some resemblance to one of the hierarchical parallel algorithms reviewed by Cantu-Paz [8].

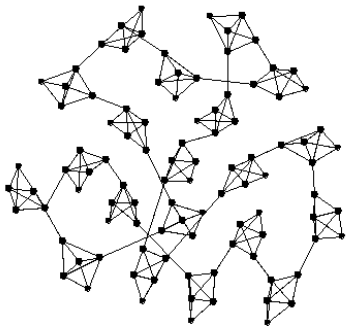


Figure 5.3: ”Connected Caveman” graph.

To generate this type of graph, we simply create fully connected clusters of vertices and then reconnect one edge in each cluster to the next cluster (the last cluster being connected to the first). A parameter, *clusterSize*, determines the size of the clusters.

Ring and Grid

The ring and grid topologies (see Figures 5.4 and 5.5 respectively) are two particular cases of *d-lattices* [40]. D-lattices are graphs in which the vertices are spatially structured and edges connect each vertex with its neighbours. The number of edges connected to a vertex is given by k. So a simple ring is a 1-lattice with $k = 2$, and a grid is a 2-lattice with $k = 4$. These two topologies are useful from an analytical point of view because their characteristic path lengths can be calculated exactly. As formulated by Watts [40], for a 1-lattice with $k \geq 2$,

$$L = \frac{n(n + k - 2)}{2k(n - 1)} \tag{5.2}$$

This shows that the length scales linearly with n; additionally, these types of graphs also generally have a low clustering coefficient, since they are spatially structured [40].

We generate ring graphs by simply connecting each vertex to the next $k/2$ vertices in two directions. Grids are generated by connecting a vertex to its $k/4$ closest neighbours in four directions. Although k is an input parameter, for this project we used $k = 2$ for rings and $k = 4$ for grids. The number of vertices for our grid graphs was required to be a square to maintain the shape.

Scale-free

For many years since random graph theory was developed by Erdos and Renyi [13], complex real-world networks were approximated by using random graphs. However, recent analysis of

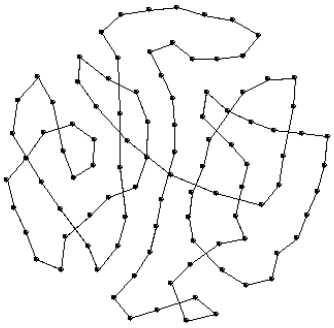


Figure 5.4: Ring graph.

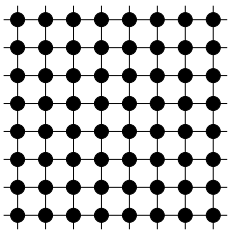


Figure 5.5: Grid graph.

large real-world systems such as the World Wide Web and social networks has shown that instead of pure random graphs, they are actually ”scale-free” [9]. In other words, the degree distribution follows a power law instead of a Poisson distribution as in random graphs [6]. Put simply, there exist a few ”central” vertices that have a much larger number of edges connected to them than the rest. In a real world example, popular web sites are likely to have a disproportionately larger number of links to them than less popular sites. Barabasi and Albert proposed that complex networks develop scale-free behaviour due to growth and preferential attachment [5]; in other words, nodes are added one at a time and have a higher probability of connecting to nodes that are already highly connected. This is a very important topology to study for the DBE because it is likely that networks that are allowed to grow on their own will develop this behaviour, since ”popular” nodes will likely receive much more attention and thus more new connections.

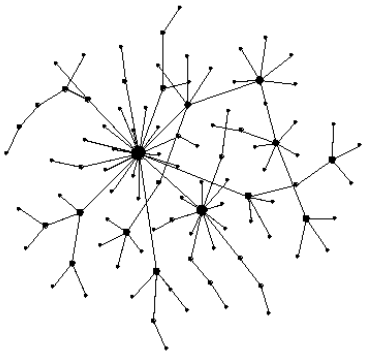


Figure 5.6: Scale-free graph.

Although many different approaches exist to generate scale-free networks ([36], [9], [11], [25]), most of them are based on the original Barabasi-Albert model [5], which we slightly modify here. Nodes are added one at a time, determining the probability $P(k_i)$ that it will connect to existing node i by,

$$P(k_i) = \frac{k_i}{\sum j k_j} \tag{5.3}$$

Our one modification places a scaling exponent b on k_i so that more or less importance can be placed on the degree distribution. Figure 5.6 shows a scale-free graph with $b = 1.5$. With b, this model can generate networks that range from something very close to a random graph (although with connections slightly biased towards nodes added at the beginning of the algorithm) to a ”hub” graph where all nodes are only connected to a single central node.

Small-world

The Kevin Bacon Game has become a well known illustration of a fascinating phenomenon found in many real-life networks [40]. In this game, an actor or actress is given a "Bacon Number" based on how closely associated with Kevin Bacon (a previously less well-known American actor) they are, with respect to films in which they play. For instance, if an actor or actress ever appeared in the same movie as Bacon, they would have a Bacon Number of 1. If they have never appeared in a film with him, but have been in a film with another actor who has, then they have a Bacon Number of 2, and so forth.

The game itself may be fun for movie enthusiasts, but the really interesting thing was the conjecture made by the game's founder that no American actor has a Bacon Number higher than 4; this conjecture was later proven using data from the Internet Movie Database (an extensive online database of films that references tens of thousands of actors and actresses) [40]. It is interesting because most actors and actresses do not appear in a very large number of films with respect to the total number of films indexed, yet they all remain a very few number of steps away from each other. This phenomenon, which also occurs in many other networks, is called the *small-world effect*.

Random graphs also show some elements of the small-world effect [32]. However, random graphs are not a very good model for social networks since they do not have a characteristically high clustering coefficient. Following Newman's definition, small-world graphs are those that have a low characteristic patch length and a high clustering coefficient [32]. Watts, in his excellent work on small-world graphs [40], devotes several chapters to their formal definition and the interested reader is encouraged to look there for more information.

For this project, small-world networks are a useful construct because they can strike a balance between keeping the number of edges connected to any one node relatively low, yet also allowing information to diffuse quickly throughout the entire network. To generate small-world networks, we used Watts' β-graph model [40] (see Figure 5.7). The algorithm begins with a perfect ring graph (generally with $k > 2$) and rewires each edge with probability β to a randomly selected vertex (disallowing self-connections). In this model, when $\beta = 0$, the graph remains a ring and when $\beta = 1$, the graph is completely rewired into a random graph. As β increases from 0, the characteristic path length quickly decreases before flattening out. For this project we chose $k = 4$ and $\beta = 0.2$, since this will give a graph with a low average degree but also a short characteristic path length.

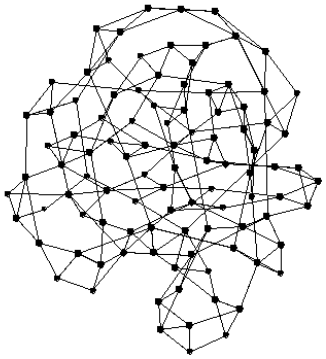


Figure 5.7: Small-world graph.

Chapter 6

NetEvo

To implement the model described in the previous chapter, we have developed NetEvo, a parallel genetic algorithm system. Since it is designed to run on a single-processor system, NetEvo is not intended to increase the actual speed of the searching, but to investigate the role that topology plays in the PGA. Since a PGA is logically hierarchical, NetEvo uses an object-oriented paradigm, and due to our existing expertise, we implemented it in C++. We also made extensive use of the Standard Template Library, especially the Vector class.

This chapter will first describe the architecture of NetEvo, then will discuss the many configuration parameters, and will conclude with an evaluation of the software.

6.1 Software Architecture

NetEvo is made up of several main object classes: Individual, EvoPatch, and EvoSim. Each EvoPatch implements a distinct GA, with Individuals living inside of it. EvoSim controls the whole simulation as well as handling migration between patches (whose topology is stored in a Graph object). A ”wrapper program” (netEvo) instantiates the EvoSim object, initialises its configuration, and begins the simulation (see Figure 6.1). NetEvo is a discrete time simulator; in other words, the simulation progresses in distinct time steps. Due to this, performance of the algorithm is measured in time steps, not real time.

netEvo

The wrapper program is responsible for loading the simulation configuration from the user, instantiating and initialising the simulation, and reporting relevant data into a human-readable format. To load configuration information, we use Richard J. Wagner’s freely-available ConfigFile class, which reads named values from a configuration file. All configuration information is stored in a config file which is loaded at run time into an EvoSimConfig object. This object is passed to an EvoSim object, and then netEvo enters into a time step loop which runs EvoSim’s

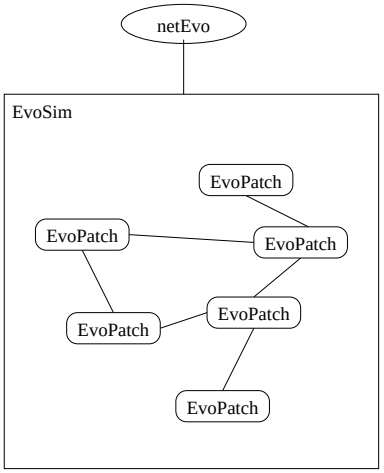


Figure 6.1: NetEvo system architecture.

update method each time step. At regular intervals, data such as average fitness and diversity is sampled. After the desired number of time steps, the EvoSim object is re-initialised and a new run is begun. If so configured, after the runs are completed the migration rate is increased and the process is begun again.

After all runs have completed, the sampled data is processed to determine the mean and standard deviation, and then outputted to the screen. The output shows the sampled time step, mean and standard deviation of the average fitness of the patches, and the mean and standard deviation of the average diversity of the patches.

EvoSim

EvoSim maintains the configuration for the simulation in an EvoSimConfig object, which gives a convenient way of passing configuration information around. EvoSim is initialised by passing in an EvoSimConfig object, which resets EvoSim's internal variables. This allows multiple runs to be conducted without re-instantiating the EvoSim object.

EvoSim also generates the list of specs which form the matrix for the SCP; this is generated once each run and is the same for all patches.

EvoSim's update() method is what drives the entire simulation; each time it is called (by netEvo), it simulates one time step. This involves iterating through all EvoPatches and running their update() methods. After running all EvoPatch updates, EvoSim iterates again through the patches, running the EvoPatch migrate() method. After retreiving the migrating individuals from each EvoPatch, EvoSim determines the neighbourhood of each patch and transfers the individuals to their destinations.

The heart of EvoSim is a Graph class; this contains not only the EvoPatches themselves, but the topology that connects them. When EvoSim is initialised, it creates the topology by using one of the network generation functions.

class Graph

Graph is a templated class that we developed for this project. In its basic form, it implements a bi-directional, weighted graph, with an arbitrary data type stored in its vertices. It allows direct access to the nodes, as well as implementing several methods that allow access to a given node's neighbours. Graph also implements several methods which allow analysis of the graph such as determining whether or not it is connected and its degree distribution.

Since for this project we used unweighted, bi-directional graphs, we implemented a child class of Graph called UBGraph, which simplified usage. We implemented several different templated functions to generate different network topologies; the UBGraph object is passed to one of these functions along with the topology parameters, and the function generates the desired topology.

This design allowed easy testing of the network generation algorithms in the beginning of the project, without having to worry about EvoSim.

class EvoPatch

Each EvoPatch object implements a complete genetic algorithm. The population, fitness evaluation, and genetic operators are all controlled at this level. Each patch is oblivious to the network of patches; each time step it only looks at its current population. A pointer to the list of specs in the EvoSim is contained inside of the patch.

EvoPatch implements a variety of genetic operators; the configuration determines which ones are run by the patch each time step. After running the operations, EvoPatch searches through its population and stores a copy of the most fit individual, if it is better than the previous fittest; this ensures that a patch will never decrease in its best fitness, even when potentially damaging genetic operations are performed.

When the migrate() method is called, EvoPatch determines which of its individuals are due to migrate. In migrate best, the best individual migrates with a probability based on migration rate; otherwise, each individual migrates with a certain probability (also the migration rate). A vector is passed into migrate() by EvoSim, and the patch simply places the individual into the vector. This prevents the patch from having to know about any of the other patches.

class Individual

The Individual class is responsible for storing the genetic representation, actually performing crossover and mutation, as well as evaluating fitness.

Since the chosen representation for this model is a bit string, we used a vector of boolean variables. This allowed us to have an arbitrary-length representation, which could even be changed at run-time. Crossover is performed by passing an individual into the crossover() method of another; this returns a new individual that can be stored where desired. To ease in usage, we overloaded some of the basic arithmetic operators for class Individual.

To evaluate fitness, the current configuration parameters that affect fitness are passed into the individual (such as the target string and penalty values). The individual evalutes fitness itself and returns the fitness value.

6.2 Configuration

There are a multitude of configuration parameters for NetEvo. Below is a list with descriptions.

Display Parameters

showRunResults	outputs the final fitness and diversity for each run
showBest	outputs the best individual in each patch at the end of the run
showUpdateFitness	outputs the current fitness each time step
showTarget	displays each target string in integer format
showSamples	outputs each time sample when it is taken
showIndSamples	outputs individual time samples

Patch Parameters

target	the target bit string, in long int format
randomTarget	generate a random target string (this overrides *target*)
targetMaxCaps	random targets will have no more than this number of enabled bits
popSize	initialise this many individuals at start
targetChangeProb	each time step the patch target will change with this probability
targetMutate	only change target by flipping bits
targetMutateFlips	when mutating target, flip this number of bits

GA Parameters

sizePenalty	the fitness penalty due to size
extraCapPenalty	the fitness penalty for each enabled bit not in the target string
missCapPenalty	the fitness penalty for each bit not in the capability string
notCoverPenalty	the fitness penalty if the individual is not a cover
crossoverType	the type of crossover performed, either tournament or elite
mutationRate	each individual mutates with this probability
mutationType	the type of mutation performed, swap or flip
flipProb	if mutationType=flip, each bit in a mutating individual flips with this probability

Topology Parameters

topology	beta, grid, ccaveman, ring, scalefree, random
topology.degree	for beta and ring, k
topology.b	for beta, β
topology.p	for scalefree, the scaling factor
topology.clusterSize	for ccaveman, the cluster size

EvoSim Parameters

numPatches	the number of patches in the simulation
specSize	the bit length of specifications
numSpecs	the number of specs generated/individual bit length
maxCaps	the maximum number of 1 bits in specs
sameTarget	set all patches to the same target
randomTargets	set all patches to different, random targets

Migration Parameters

migrationRate	the probablity of individuals migrating
copyMigrate	copy individuals instead of moving them
migrateBest	only migrate the best individual in a patch
copyToAllNeighbours	if copyMigrate, copy to all neighbours
varyMigrationRate	run multiple simulations
migrateRateStepSize	the amount to vary each simulation
migrationRateSteps	how many simulations to run

Simulation Parameters

numRuns	how many runs to make each simulation
timeSteps	how many time steps per run
numTimeSamples	how many samples to take during a run
rawFitness	sample raw fitness without penalties applied

6.3 Evaluation

The NetEvo system was easy to use for this project and provided an intuitive way to change network topologies and simulation parameters. While an attempt was made to implement the algorithm as efficiently as possible, efficiency was not the primary goal; however, in practical terms the system performed fine and did not present any problems.

Implementing the simulator as an object-oriented system proved to be an excellent choice since it allowed very easy reconfiguration of the simulation and maintained an intuitive structure. Storing configuration information first in a configuration file and then internally inside of an object turned out to save vast amounts of time and allowed the simulation to be run multiple times without reinstantiating the object.

Chapter 7

Experiments

Using NetEvo, we ran several experiments to observe the PGA and the flow of information around the patches. The first set focused on single patch behaviour, in particular attempting to arrive at a set of parameter values that could be used for the rest of the experiments. The second set looked at both the diversity and fitness of patches over time with respect to different network topologies.

Methodology

All experimental runs were repeated 30 times for statistical viability. We collected the mean and standard deviation and when appropriate these are shown on the graphs. T-tests were usually not performed since the results were generally clear.

For all experiments, we tried to set the parameters to reasonable values; this was often tricky since there were so many parameters. The rest were set to the best guess values in the context of the experiment. Many of the parameters have a dramatic effect on the model; in particular, parameters such as specSize, numSpecs, and maxCaps can fundamentally change the search space. Within the time constraints of this project, however, we chose values that were the best balance between realistic experimental times and reasonable difficulty.

Unless specified otherwise, for each run, each patch was assigned a newly-generated random target requirement bit string.

7.1 Single Patch

Since there are a large number of variable parameters in the model, the first set of experiments focused on a single patch to determine optimal parameter values.

Set	Size	ExtraCap	MissCap	NotCover
1	1	1	1	0
2	1	1	1	10
3	1	1	5	50
4	1	5	10	50
5	1	1	2	10

Table 7.1: Penalty parameter sets.

7.1.1 Fitness Parameters

One of the most important decisions was the assignment of fitness penalty parameters. While testing these parameter sets, we used the ”raw” fitness which was the sum of the number of specifications in the set, the number of excess capabilities, and the number of unmatched capabilities. Since the ideal solution would consist of exactly one specification which matched the target exactly, smaller fitnesses were better. While the raw fitness (the equivalent of setting all penalty parameters to 1) works, we wanted to see if alternate assignments would improve the quality or discovery speed of solutions. Additionally, it is most important to actually have all capabilities matched, so that should have a higher weighting.

We tested five different parameter sets (see Table 7.1) to see if there was a significant difference in solution quality or discovery speed. Set 1 is the baseline, equivalent to the raw fitness. Set 2 imposes a significant penalty if the individual is not in fact a cover which is intended to weed out the individuals that are not covers. Set 3 is designed to put extra pressure on the individuals by both putting a severe penalty on individuals that are not covers, as well as a higher penalty for not meeting capabilities. Set 4 continues this trend by increasing the penalties except for size. Set 5 is a modification of set 2 that is designed to give slightly more pressure to match capabilities. Figure 7.1 shows the raw fitness over time for the five penalty sets.

Analysis

At first glance, these results may seem confusing; the fitnesses of penalty sets 2 and 5 actually increase at the beginning before converging to the same shape as the other sets. The reason for this is that this graph charts the raw fitness, which does not take into account whether or not the individuals are a cover or not. Since penalty sets 2 and 5 have similar penalties for size, extra capabilities, and missed capabilities, it takes longer to find a set that is actually a cover. For all but penalty set 1, by the end of the run the best individual had found a cover. In any case, there was not a significant difference between the penalty sets after sufficient time had elapsed.

Eventually the penalty parameter selection must be determined by the business model used in the DBE; for instance, different software specifications might have different costs based on a variety of factors. For the purposes of this project, we used penalty set 3 for most experiments since it is assumed that it is most important to ensure a cover of all the required capabilities,

while size and extra capabilities are not as important (but still ideally should be minimised in the long run).

7.1.2 Maximum Specification Capabilities

The next experiment was to determine the effect of changing the maxCaps parameter. It seems likely that software specifications in the DBE will only enable a small number of capabilities, so this parameter should probably be set to something small, e.g. 3 or 4. However, a range of values were tested to examine the behaviour of the GA (see Figure 7.2).

Analysis

For maxCaps greater than 1, there was not a large difference in terms of fitness quality, although in general the final fitness was better for larger values of maxCaps.

Again, the business model and environment of the DBE will influence the final behaviour of this parameter. Since it makes sense that software providers would often provide a few capabilities (i.e., more than one) but rarely a large number, we chose maxCaps=4 for the rest of the experiments.

7.1.3 Population Size

Another important parameter to set was the patch population size. Smaller population sizes tend to converge faster, but may not arrive at solutions that are as good quality as populations with larger sizes [8]. However, with a larger size populations usually take much longer to converge and also take more processing power. We tested a variety of population sizes in an attempt to find an optimal value (see Figure 7.3).

Analysis

While a population size of 20 converges faster, its final fitness value is not optimal. While the larger sizes usually converge eventually, they take much longer to do so. According to this experiment, a population size of 50 appears to be optimal since it is a good balance between convergence speed and finding better fitness values; for the rest of the experiments, the patch population size was set at 50.

7.1.4 Number of Specifications

The number of possible software specifications also has a dramatic effect on the algorithm as a whole, as this essentially determines the space in which the GA will search. Since this project is interested in the effects of different network topologies, it was important to find a parameter value that was a good balance between difficulty and speed. Since the length of the individual's

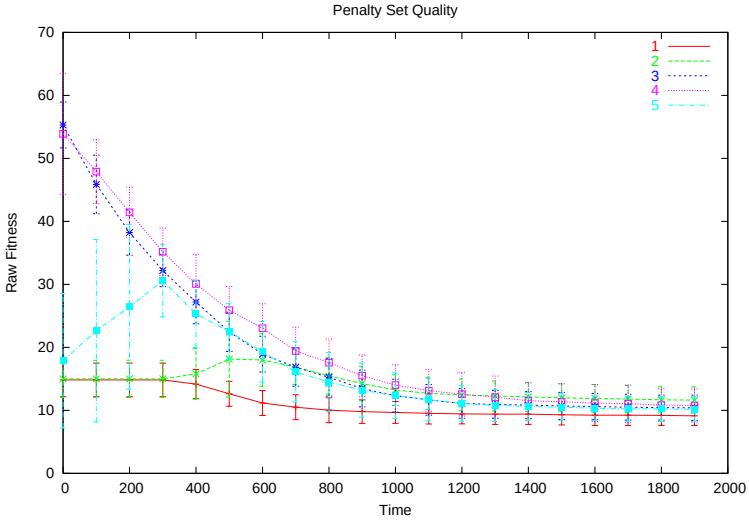


Figure 7.1: Penalty set quality comparison.

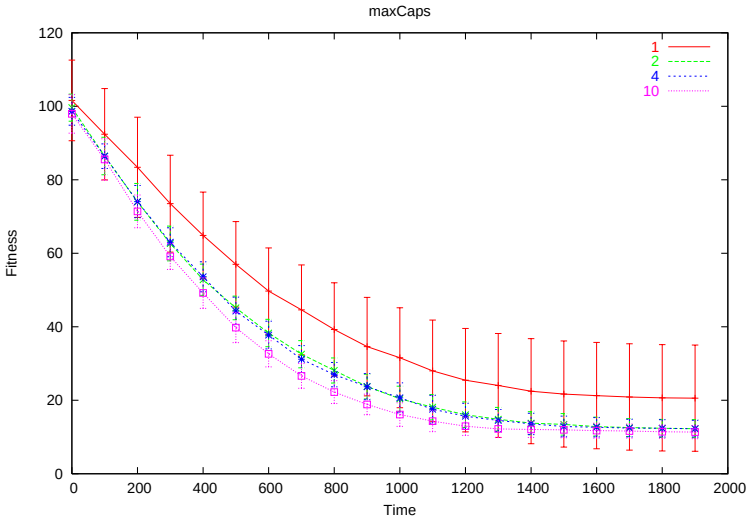


Figure 7.2: maxCaps.

bit string is the number of specifications, too large a value can quickly make the algorithm impractical to run. Figure 7.4 shows a range of different values for numSpecs and their fitnesses over time.

Analysis

Since specifications are generated randomly, for numSpecs less than 50, there would often not be enough specifications with the right bits to create proper covering sets. On the other hand, for numSpecs greater than 500, it was too hard for the algorithm (with the existing parameters) to find a decent solution and so it would usually converge to a degenerate solution (usually the empty set), since this at least minimises the size. As a balance between difficulty and time, we chose 200 as the default number of specifications. This choice fundamentally affected the rest of the project, and perhaps this value was too low; further research should investigate this more fully.

7.1.5 Crossover Type

We next tested the two selection/crossover operators (elite and binary tournament) for both fitness (Figure 7.5) and diversity maintenance (Figure 7.6). While it is important to converge on a good fitness value, when diversity is too low it can prevent novel solutions from being created.

Analysis

Elite crossover, as expected, converged more quickly than tournament crossover. However, tournament crossover did arrive at the same fitness (or better) eventually, and maintained diversity much longer. For this reason, we chose binary tournament to be the crossover operator used in the rest of the experiments.

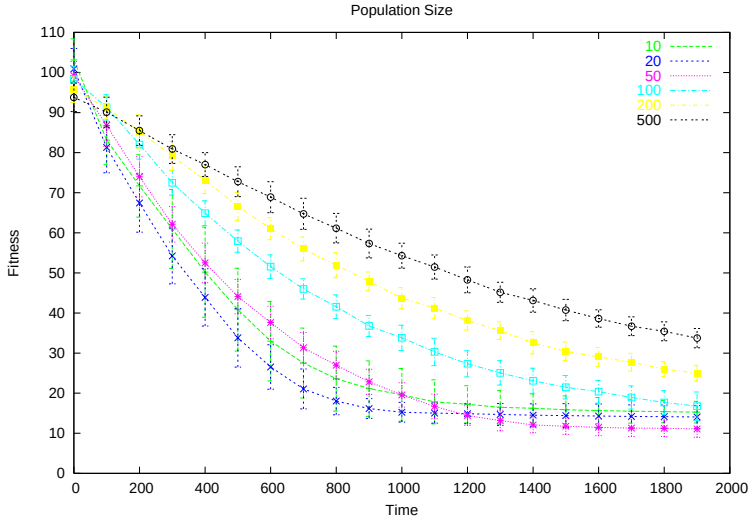


Figure 7.3: Population size.

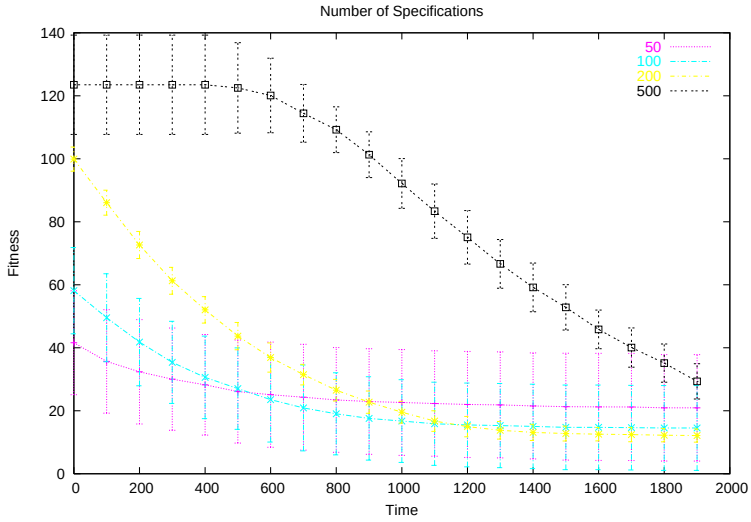


Figure 7.4: Number of Specifications.

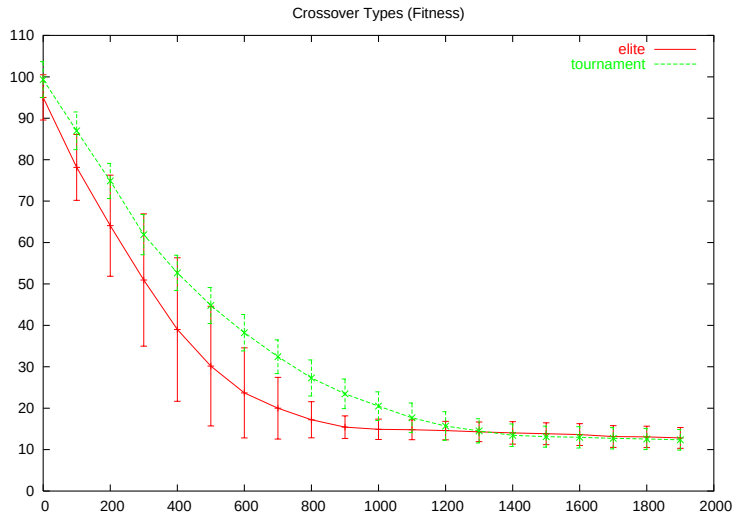


Figure 7.5: Crossover types: fitness vs time

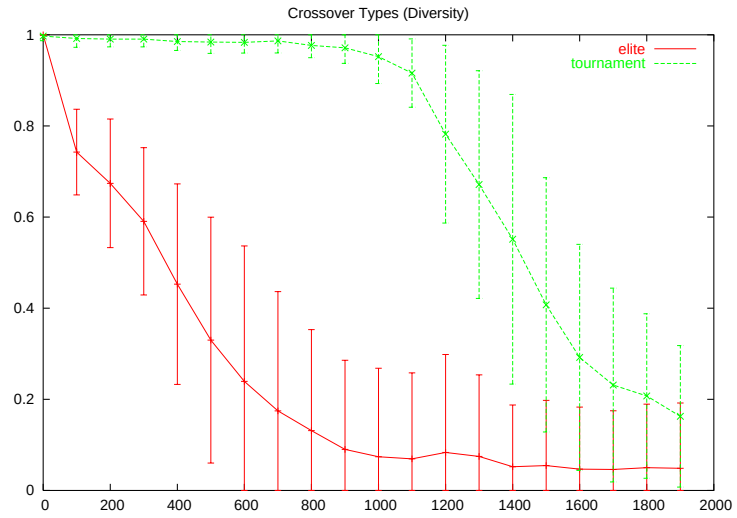


Figure 7.6: Crossover types: diversity vs time.

7.2 Multiple Patches

This set of experiments looked at the relationship between network topology and the diversity and fitness of the patches.

The diversity of a patch was measured by counting the number of different individuals in the patch divided by the patch population size. This arrives at a number between 0 and 1, where 0 represents total homogeneity and 1 total heterogeneity. The global diversity was not measured since it would be unclear what it meant; for instance, in a two patch system if both patches were completely homogenous but different from each other, this would be indistinguishable from a situation in which both patches were half and half.

Diversity is important to observe because, since crossover is the main searching operator, a less diverse population will reduce the number of directions being explored. However, when converging on good solutions it is often necessary to have more individuals close to the good solution to avoid creating an offspring that is too unfit; this balance can be difficult to acheive.

7.2.1 Same Targets

The first part of this set of experiments uses the typical PGA setup, with all patches having the same target bit string. The intention is that since all patches are all trying to solve the same problem, if patches are able to broadcast their good solutions then that information will spread throughout the network, which will result in good individuals being found sooner. We used binary tournament selection, uniform crossover, swap mutation, and all networks consisted of 25 patches.

While the mutation operator exists to introduce diversity into the populations, we wanted to see how the diversity was also affected by the migration of good individuals. Since diffusing to all neighbours worked much faster in the diffusion model, when a patch is selected to migrate it copies its best individual to all of its neighbours, overwriting the worst individual in the target patches. Figures 7.7 and 7.8 show the diversity over time for migration rates of 0.00 and 0.05 respectively, while Figures 7.9 and 7.10 show the fitness over time for the same migration rates.

Analysis

At m=0.00, there is no migration so there is no difference between the different topologies; the diversity decreases steadily due to crossover, whilst the fitness steadily improves. By m=0.05, the small-world topology loses diversity very quickly and reaches equilibrium at a lower level than the rest of the topologies. The equilibrium diversity is non-zero because of the fixed mutation rate; with no mutation the population would keep reproducing until it was completely homogenous. The fitness of the small-world network also improves quicker than the other topologies, although it converges on the same fitness.

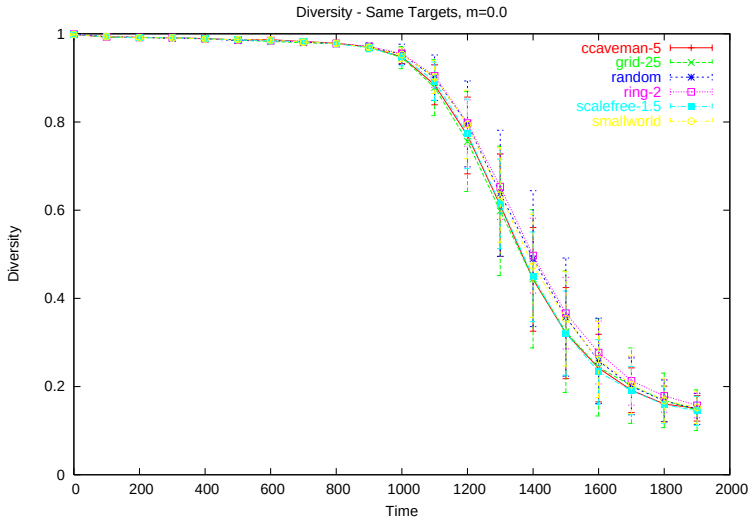


Figure 7.7: Diversity: same targets, migration rate = 0.00

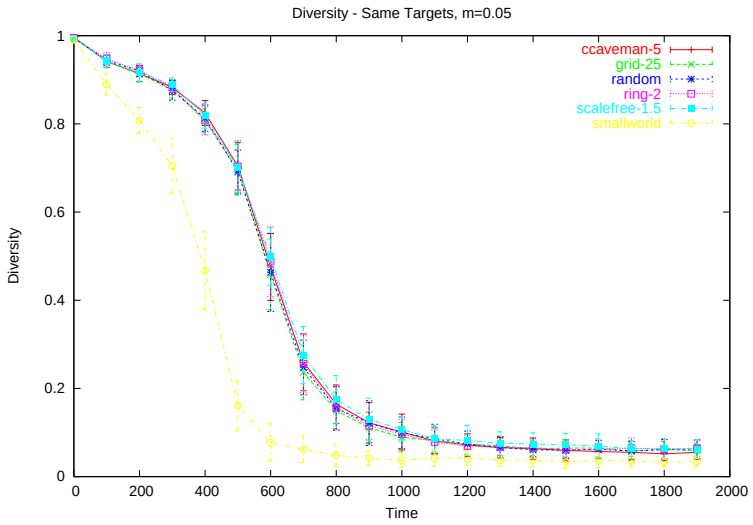


Figure 7.8: Diversity: same targets, migration rate = 0.05. The small-world topology loses diversity much faster than the rest of the topologies.

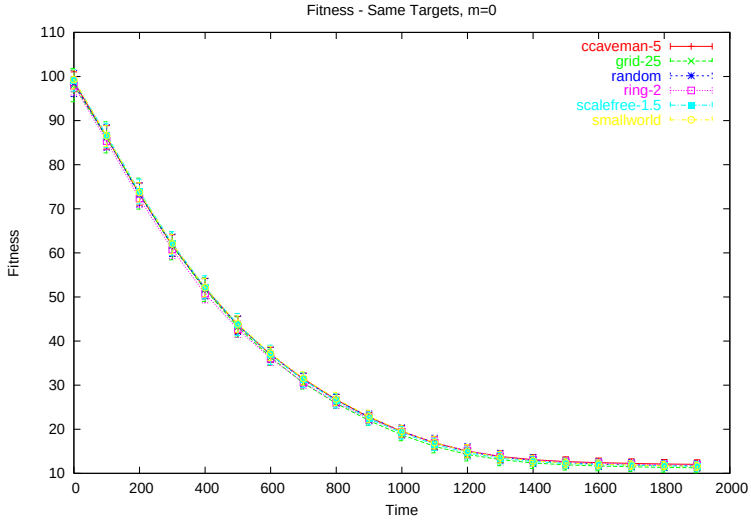


Figure 7.9: Fitness: same targets, migration rate = 0.00

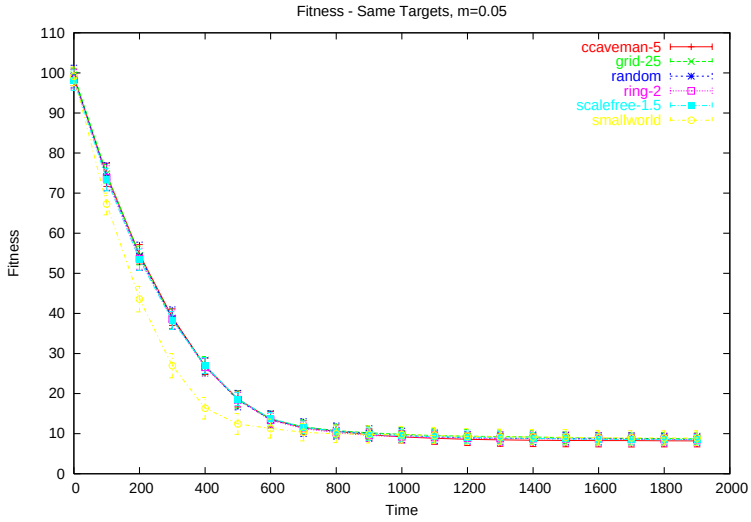


Figure 7.10: Fitness: same targets, migration rate = 0.05

The likely reason for the small-world topology losing diversity so quickly is that since the clustering coefficient is high, good individuals that are passed to neighbours are likely to be sent to a whole cluster, and the next time step those same individuals will likely be passed back. Since the incoming individual is copied over the worst individual in the target patch, inside a cluster good individuals will quickly bounce around, replacing bad individuals until there are no bad individuals left. Since the small-world network also has a short characteristic path length, good individuals will also very quickly diffuse throughout the entire network. This early convergence translates into increased fitness until it can no longer improve.

Figures 7.11 and 7.12 show the diversity and fitness over time in the small-world topology for several different migration rates. The higher the migration rate, the sooner the diversity drops; this could be problematic if it converges to a suboptimal solution, although in this case the fitness converged very quickly on the eventual found solution.

7.2.2 Different Targets

In the next experiment, each patch was given a different, randomly generated target bit string. The main ramification of this is that the best individual for one patch is likely to be very unsuitable for a different patch. It is even likely that it will be worse than the individual that it is replacing (although this will already be the worst in the destination patch). However, this migration can possibly create new diversity in patches that might have already converged on a suboptimal solution. It might have a slight drawback on the short-term fitness of the target patch, however. This setup is meant to be closer to what the DBE framework will encounter, and attempts to see how well the PGA works in this context. Figures 7.13, 7.14, 7.15, and 7.16 show the diversity and fitness over time for m=0.00 and m=0.05.

Analysis

All of the topologies show a similar shaped diversity curve: at first diversity decreases, then slowly increases before dropping off dramatically and converging to its equilibrium. The reason for this shape is not completely clear, although since the small-world topology again drops in diversity sooner, we suspect that it is caused by the propagation of individuals that are a local minimum of fitness. For instance, given the fitness parameters chosen, an individual bit string of all 1s will definitely be a cover and will not have any missing capabilities. Since this type of individual will be successful (at least in the short run) in any patch, it is quickly propagated around; as before, in the small-world topology it will quickly bounce back and forth over-writing less fit individuals, which is probably why that topology loses diversity much quicker. The diversity peak in the middle is probably caused by the migration of other nodes; eventually better solutions are found and then crossover begins to converge.

As in the previous experiment, the small-world topology is able to find higher quality individuals faster, although it too converges eventually to the same fitness as the other topologies.

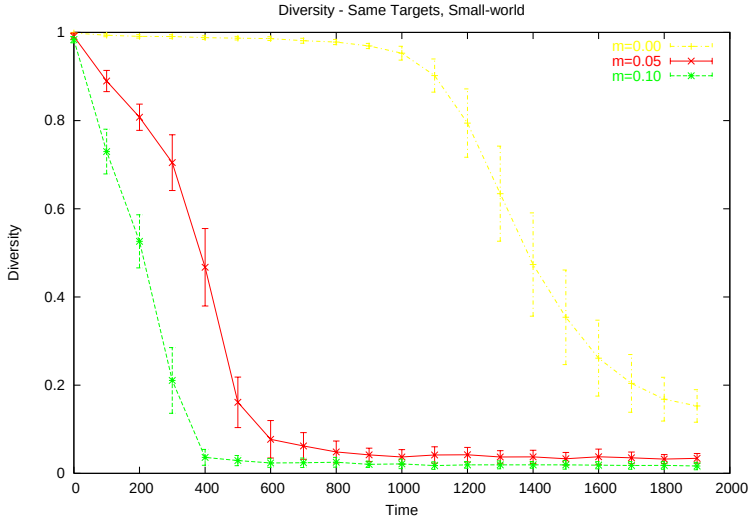


Figure 7.11: Diversity: same targets, small-world topology.

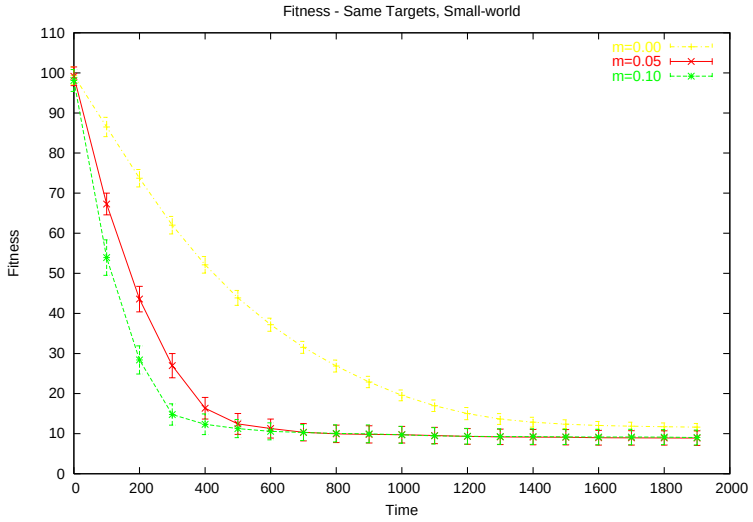


Figure 7.12: Fitness: same targets, small-world topology.

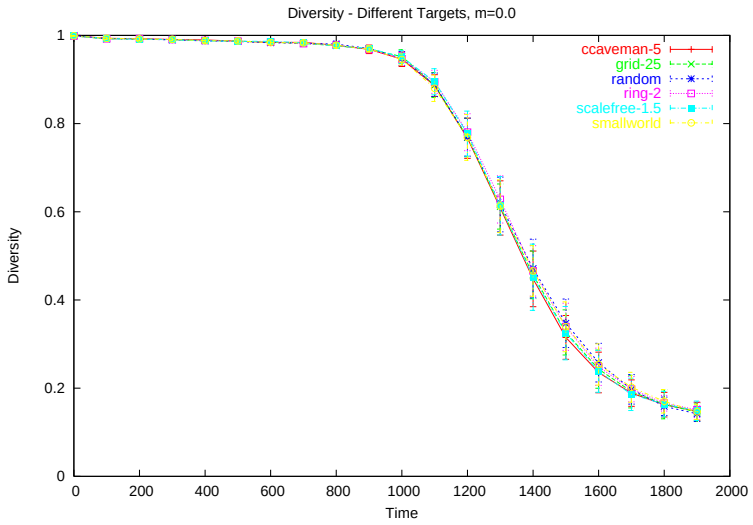


Figure 7.13: Diversity: different targets, migration rate = 0.00

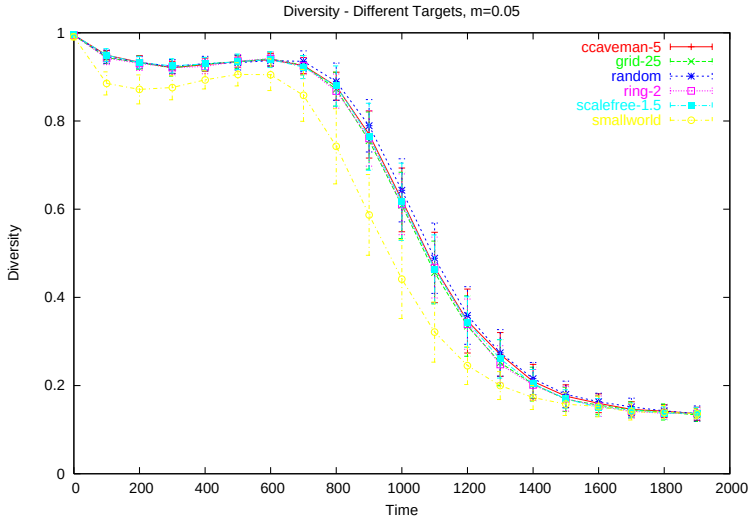


Figure 7.14: Diversity: different targets, migration rate = 0.05

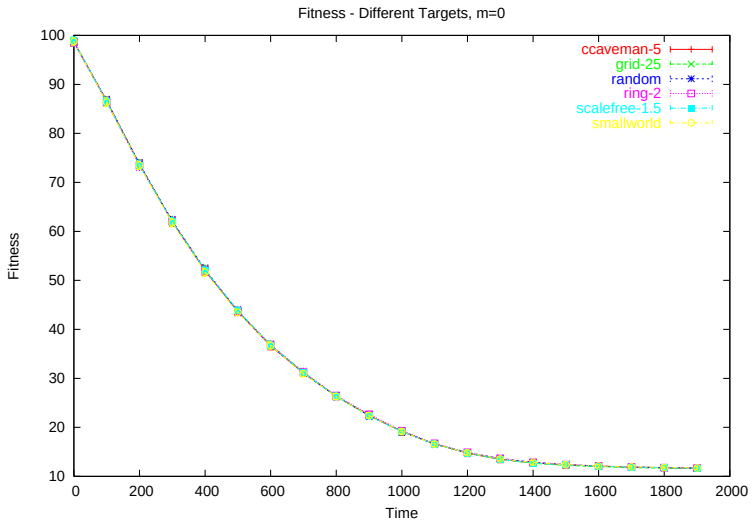


Figure 7.15: Fitness: different targets, migration rate = 0.00

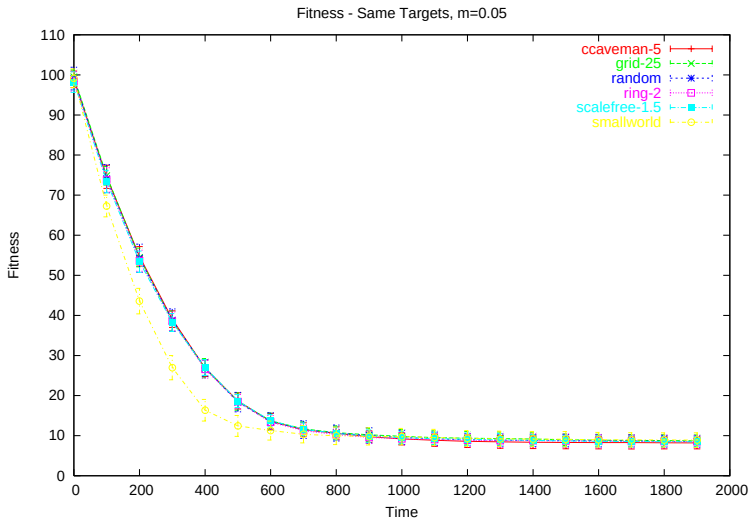


Figure 7.16: Fitness: different targets, migration rate = 0.05

7.2.3 Changing Targets

Diverging more and more from the traditional experimental domain of PGAs, our next set of experiments studied the effect of the PGA when targets are occasionally changed. This is a potentially more likely real-world scenario, since SMEs might change their requirements from time to time. We were most interested to see what the effect that migration had on this problem. Since it is more likely for SMEs to only slightly change their requirements instead of creating an entirely new one, targets were mutated instead of being generated anew. We found that, as before, the small-world network performed better than the other topologies. Figures 7.17 and 7.18 show the diversity and fitness over time for the small-world graph. In these experiments, the target mutation rate was set at 0.025 and would flip 2 bits when mutating.

Analysis

For this problem, it is very important for patches to maintain at least some diversity; if a population has converged on a good solution for its target string, if the target then suddenly changes the population will take longer to find another solution path. The results of this experiment show that non-zero migration rates maintain a better fitness level across the patches then without migration. Although the diversity decreases initially for the small-world graphs, it converges at a higher diversity than with no migration, which is beneficial for when the targets change. The fitness curve decreases very quickly with a migration rate of 0.1; this is likely also caused by the propagation of individuals that are very general (i.e., all 1s). As soon as a patch discovers one, it is quickly propagated throughout the network, and in fact for many patches it probably remains the best solution.

7.2.4 Results

While it was surprising that there was not more differentiation in the other network topologies, the interesting result was the superior performance of the β model small-world topology. These results suggest that it would be a good choice for connecting nodes together in the DBE since it allows information to quickly propagate due to its low characteristic path length and high clustering coefficient. However, when using a model similar to ours, it does not appear that there is very much difference in the other topologies.

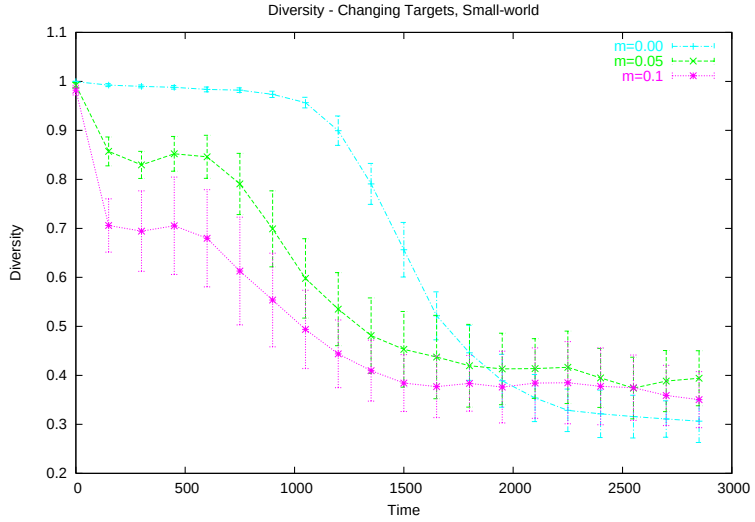


Figure 7.17: Diversity: changing targets, small-world topology.

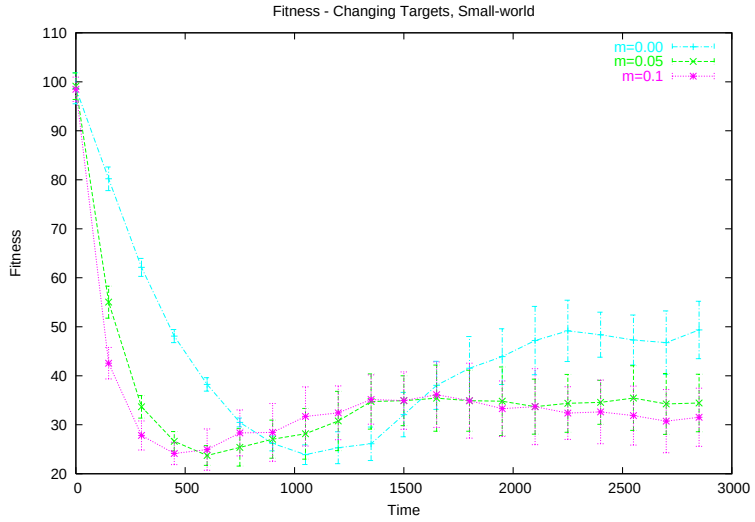


Figure 7.18: Fitness: changing targets, small-world topology.

Chapter 8

Conclusions

This paper has presented a parallel genetic algorithm model to solve a modified version of the set covering problem, as well as a software package that implements it. We ran a range of experiments that both calibrated the GA as well as studied the effect of different network topologies on information diffusion and PGA performance.

We found that the small-world graph performed the best in our simulation runs in terms of fitness and information propagation. This suggests that the β model would be a good choice to implement a network that required information to diffuse quickly throughout the network. For the other topologies, we found that there was not a significant difference in either fitness or diversity. Small-world networks also seem to have advantages in terms of propagation time, although this might only pay off as the size of the network grows sufficiently large.

The results obtained here are by no means exhaustive; there remain a large number of possible directions to go from this project. The theoretical basis for GAs and some types of PGAs are relatively well understood. However, the multiple-deme PGA model can present some problems to analysis [8]. We are currently pushing forward our analytical models of propagation time in networks, in collaboration with Dr. Leslie Goldberg of the University of Warwick, and are making considerable progress on verifying our conjectures.

One avenue of future work would be to dramatically increase the difficulty of the problem being solved; the limited number of specs to search through created a rather easy problem for the GA to solve, so the parallel effects were not necessarily needed. Typical PGAs are solving SCP problems of up to 500 rows and 5000 columns [37], which is quite a bit larger than the problem solved here. In fact, a real world DBE framework would probably not use as simple a capability model as a bit string but instead some sort of BML; this would most likely mean a rather different problem that needed to be solved.

More work should be done observing the effect of the β parameter on the small-world networks; our chosen value of 0.3 was a best-guess value and was successful, but a wider range should be tried to find the specific point at which the small-world effect begins to work. Although

the β graph model is popular, it is not the only generation model that displays small-world effects [32]; other generation models should be tried to see if the same results are observed.

It is not clear why there was not more differentiation between the other topologies; future work should investigate this phenomenon to find its cause.

Other graph models should be tried, such as the extreme case of a scale-free graph where all vertices are connected to one central node. This would be similar to the master-slave PGA topology. Another interesting topology to test would be a bus model, where there is a long chain of nodes with a small number of secondary nodes connected to the central vertices. Finally, a fully connected graph should provide an extreme, limiting case for experimental testing.

Larger graphs should also be investigated; our graph sizes were limited due to performance considerations; larger graphs should provide a more accurate simulation of the underlying dynamics. Since the path length in ring and grid models scales linearly, there might be more differentiation in PGA performance between the topologies when the number of patches is dramatically larger.

NetEvo proved to be very useful for these experiments and there is a very large range of experiments that it is capable of performing. The most likely direction of future work on NetEvo should go towards an efficiency audit of the code, since efficiency was not the primary concern whilst implementing it. A faster NetEvo would enable more experiments to be run in a more practical time-frame.

The actual business model of the DBE, when it is implemented, will dictate more specifically how feasible our model actually is. Since the model here is relatively simple, it is difficult to say how applicable these results will be to the DBE; future work would be dramatically influenced by the direction the DBE itself takes.

Bibliography

[1] R. Anderson and R. May. *Infectious Diseases of Humans: Dynamics and Control*. Oxford UP, London, 1991.

[2] R.M. Anderson. *Population dynamics of infectious diseases: theory and applications*, chapter Directly transmitted viral and bacterial infections of man. Chapman and Hall, London, 1982.

[3] N. Bailey. *The Mathematical Theory of Infectious Diseases*. Charles Griffin, London, 1975.

[4] Kennon Ballou. Bugs: a spatially-explicit individual-based metapopulation epidemic simulator. Mini-project report, University of Birmingham: School of Computer Science, 2004.

[5] A.L. Barabasi and R. Albert. Emergence of scaling in random networks. *Science*, 286:509–512, 1999.

[6] A.L. Barabasi and R. Albert. Statistical mechanics of complex networks. *Rev. Mod. Physics*, 74:47–97, 2002.

[7] J.E. Beasley and P.C. Chu. A genetic algorithm for the set covering problem. *European Journal of Operational Research*, 94:392–404, 1996.

[8] Erick Cantu-Paz. A survey of parallel genetic algorithms. Technical report, University of Illinois at Urbana-Champaign, 1998.

[9] Qinghua Chen and Dinghua Shi. The modeling of scale-free networks. *Physica A*, 2004.

[10] H. Chernoff. Asymptotic efficiency for tests based on the sum of observations. *Annals of Mathematical Statistics*, 23:493–507, 1952.

[11] Chavdar Dangalchev. Generation models for scale-free networks. *Physica A*, pages 659–671, 2004.

[12] C. Darwin. *On the Origin of Species*. John Murray, 1859.

[13] P. Erdos and A. Renyi. On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci.*, 5:17–61, 1960.

[14] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman, 1979.

[15] D.E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Reading, MA., 1989.

[16] W.H. Hamer. Epidemic disease in england. *The Lancet*, 1:733–739, 1906.

[17] I Hanski. Single-species metapopulation dynamics: concepts, models, and observations. *Biological Journal of the Linnean Society*, 42:17–38, 1991.

[18] I. Hanski. *Metapopulation Ecology*. Oxford UP, Oxford, 1999.

[19] I Hanski and M Gilpin. Metapopulation dynamics: brief history and conceptual domain. *Biological Journal of the Linnean Society*, 42:3–16, 1991.

[20] I. Hanski and M.E. Gilpin. *Metapopulation Biology*. Academic Press, San Diego, 1997.

[21] S. Harrison. Local extinction in metapopulations. *Biological Journal of the Linnean Society*, 42:73–88, 1991.

[22] W. Hoeffding. Probability for sums of bounded random variables. *Journal of American Statistical Association*, 58:13–30, 1963.

[23] J.H. Holland. *Adaptation in Natural and Artificial Systems*. MIT Press, 1975.

[24] J.H. Holland. Genetic algorithms. *Scientific American*, 278(1):66–72, 1992.

[25] Konstantin Klemm and Victor M Eguiluz. Growing scale-free networks with small-world behavior. *Physical Review E*, 65, 2002.

[26] Marcelo Kuperman and Guillermo Abramson. Small world effect in an epidemiological model. *Physical Review Letters*, 86(13), 2001.

[27] R. Levins. Some demographic and genetic consequences of environmental heterogeneity for biological control. *Bulletin of the Entomological Society of America*, 15:237–240, 1969.

[28] Z. Michalewicz. *Genetic algorithms + Data structures = evolution programs*. Springer-Verlag, 1996.

[29] M. Mitchell. *An Introduction to Genetic Algorithms*. MIT Press, 1998.

[30] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, New York, 1995.

[31] Francesco Nachira and et. al. Towards a network of digital business ecosystems fostering the local development. Discussion Paper, Digital Business Ecoystem Project, September 2002.

[32] M.E.J. Newman. Models of the small world. Technical report, Sante Fe Institute, 2000.

[33] A. Okubo. Diffusion and ecological problems: Mathematical models. In K. Krickeberg, editor, *Biomathematics*, volume 10. Springer-Verlag, 1980.

[34] Thomas Power and George Jerjian. *Ecosystem: Living the 12 Principles of Networked Business*. Prentice Hall, 2001.

[35] Michael Rothschild. *Bionomics*. Henry Holt, 1990.

[36] Jari Saramaki and Kimmo Kaski. Scale-free networks generated by random walkers. *Physica A*, 2004.

[37] M. Solar, V. Parada, and R. Urrutia. A parallel genetic algorithm to solve the set covering problem. *Computers and Operations Research*, 29:1221–1235, 2002.

[38] Don Tapscott, David Ticoll, and Alex Lowy. *Digital Capital: Harnessing the Power of Business Webs*. Harvard Business School Press, 2000.

[39] J.H. Vandermeer and D.E. Goldberg. *Population ecology: First Principles*. Princeton University Press, 2003.

[40] Duncan J. Watts. *Small Worlds: The Dynamics of Networks between Order and Randomness*. Princeton University Press, 1999.

[41] R.J. Wilson and J.J. Watkins. *Graphs: An Introductory Approach*. Wiley, New York, 1990.

[42] Damian H. Zanette. Critical behavior of propagation on small-world networks. *Physical Review E*, 64, 2001.