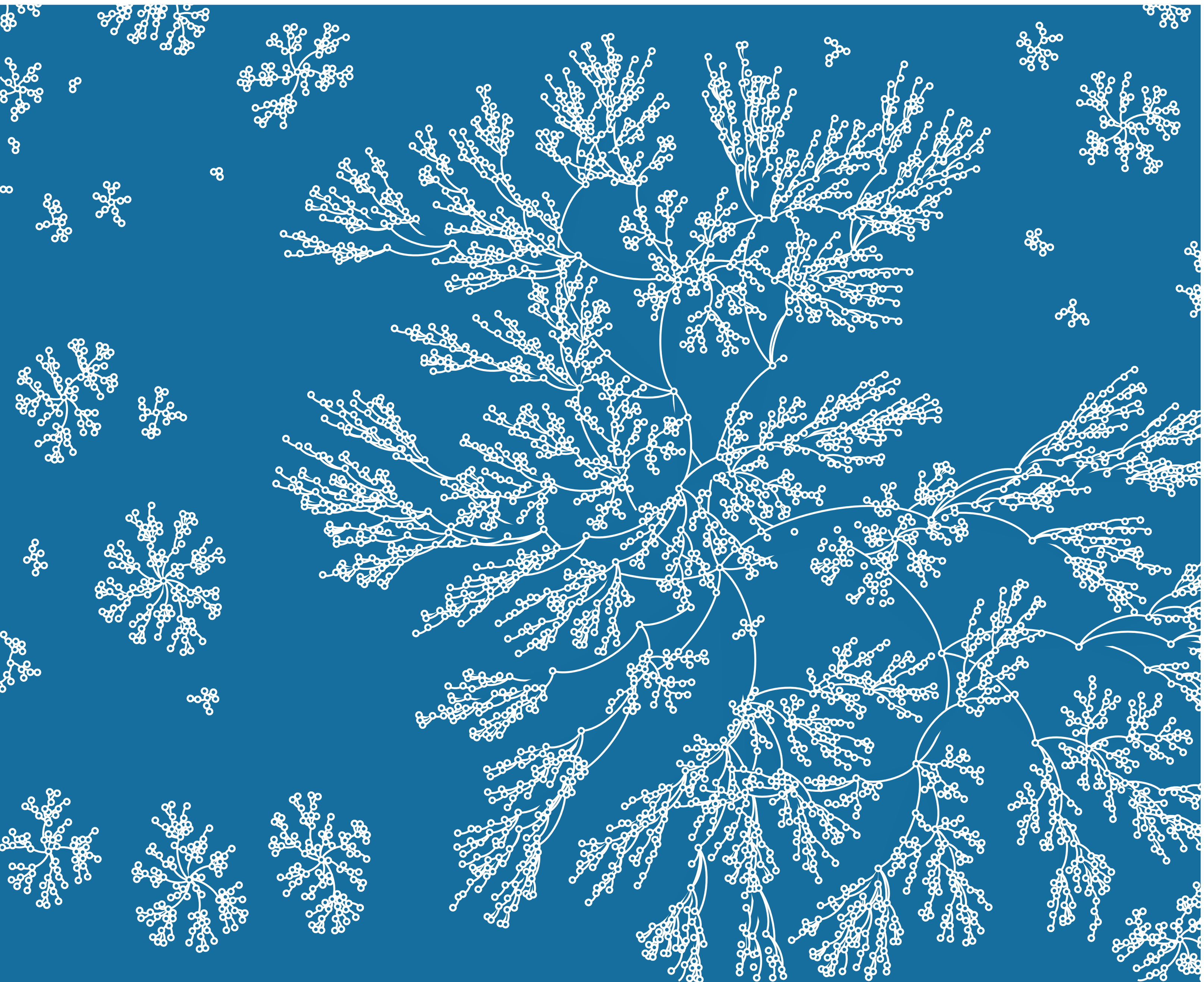
# SIGEVOlution

newsletter of the ACM Special Interest Group on Genetic and Evolutionary Computation

Volume 6
Issue 1



## in this issue

# Editorial

I was not able to attend GECCO this year, but everybody told me it was as great as usual and I am sure that all the people who attended it enjoyed it a lot! I had not spent the second week of July in Milan since 2000, when I could not attend GECCO-2000 in Las Vegas. Thus, spending the GECCO week at work while all my friends were enjoying themselves in Philly, has been kind of weird. Indeed, I already checked the flights to Amsterdam and I plan to book a ticket soon. So I guess I will see you at GECCO-2013!

Today, we start a brand new SIGEVOlution volume with a very nice cover and two even nicer articles. One, by Soo Ling Lim and Peter J. Bentley, presents AppEco an artificial life framework that has been applied to the modelling of a very well-known ecosystem, the Apple's App Store. If you want to become a successful developer for any mobile market, you should definitively read this article. AppEco appeared in the April issue of the New Scientist (check the article here). The article published here is a reprint of the paper presented at GECCO-2012 in July, with the permission of ACM. The next paper by William B. Langdon describes an application of genetic programming to the long-term outcome prediction of breast cancer using Emerald, a supercomputer containing 1008 x86 CPU cores and 372 nVidia M2090 Tesla GPUs. That's massive parallel EC!

The cover shows a graph from a study regarding the effect of app publicity using AppEco that Soo Ling and Peter recently published in ALife XIII. Consider it an appetizer of what you might want to read after you are done with the papers in this issue.

At the end, my due thanks go to the authors, Soo Ling Lim, Peter J. Bentley and William Langdon, the board members, Dave Davis and Martin Pelikan, and to the friends who help me in this adventure.

Pier Luca
July 25, 2012

[illegible]

**SIGEVOlution Volume 6, Issue 1**

Newsletter of the ACM Special Interest Group on Genetic and Evolutionary Computation.

**SIGEVO Officers**

Wolfgang Banzhaf, Chair
Una-May O'Reilly, Vice Chair
Marc Schoenauer, Secretary
Franz Rothlauf, Treasurer

**SIGEVOlution Board**

Pier Luca Lanzi (EIC)
Lawrence "David" Davis
Martin Pelikan

**Contributors to this Issue**

Soo Ling Lim
Peter J. Bentley
William B. Langdon

**Contents**

ISSN: 1931-8499

# How to be a Successful App Developer: Lessons from the Simulation of an App Ecosystem

Soo Ling Lim, Department of Statistical Science, University College London, s.lim@cs.ucl.ac.uk
Peter J. Bentley, Department of Computer Science, University College London, p.bentley@cs.ucl.ac.uk

App developers are constantly competing against each other to win more downloads for their apps. With hundreds of thousands of apps in these online stores, what strategy should a developer use to be successful? Should they innovate, make many similar apps, optimise their own apps or just copy the apps of others? Looking more deeply, how does a complex app ecosystem perform when developers choose to use different strategies? This paper investigates these questions using AppEco, the first Artificial Life model of mobile application ecosystems. In AppEco, developer agents build and upload apps to the app store; user agents browse the store and download the apps. A distinguishing feature of AppEco is the explicit modelling of apps as artefacts. In this work we use AppEco to simulate Apple's iOS app ecosystem and investigate common developer strategies, evaluating them in terms of downloads received, app diversity, and adoption rate.

## 1 Introduction

It pays to be an app developer. Some of the world's most recent millionaires made their money from mobile apps. For example, Ethan Nicholas made his million from his iShoot app in less than a year [1]. Rovio, the developer of Angry Birds, made a revenue of $100 million in 2011 [2]. The revenue generated from app sales is estimated to surpass $15 billion in 2011 and reach $58 billion by 2014 [3]. However, not all app developers are so lucky. In fact, the majority of developers make little or no profit from their apps. Reports suggested that 80% of paid apps in the Android Market have been downloaded less than 100 times [4].

While such problems may be familiar to those in the music or publishing industries, app ecosystems (comprising developers, users, and apps) face challenges that are brand new to the software industry. App store owners face the challenges of presenting the rapidly increasing app store content to the users and encouraging users to download apps. App users have difficulty in finding good apps amongst the vast number of alternatives. App developers find it increasingly difficult to make their apps stand out among hundreds of thousands of other apps in the app store, achieve downloads, and make profit.

Competition is so fierce and advertising space so congested that the question of how to be a successful app developer is now on the lips of thousands of programmers around the world. But which strategy is best? One approach to answering this question might be to experiment with a real app store: flood the store with thousands of new apps developed using specific strategies and measure their success. However, some strategies, such as copying the apps of others, are difficult to try in the real world. One developer faced a $12.5 million lawsuit for allegedly copying a $3 beer-drinking novelty app that allows users to virtually drink a pint by tilting their iPhone[1]. Consequently for this work we present an Artificial Life (Alife) agent-based model as an experimental tool to address such questions. Alife methods have proven their worth with many previous simulations of ecosystems.

[1] http://www.wired.com/gadgetlab/2008/10/indie-iphone-de/

In this paper, we present AppEco, a model of app ecosystems. AppEco models developers (agents that build apps) and users (agents that download apps). It simulates the app store environment, which hosts and organises content created by the developers, and enables users to browse and download apps.

Significantly, AppEco also models apps – artefacts produced by the developers and downloaded by users – and their features. AppEco allows us to conduct experiments, test hypothesis about various processes in the ecosystem, and ask “what if” questions, all of which are difficult if not impossible to conduct in a real-world setting. Here, we use AppEco to simulate Apple’s iOS app ecosystem and investigate common strategies adopted by developers.

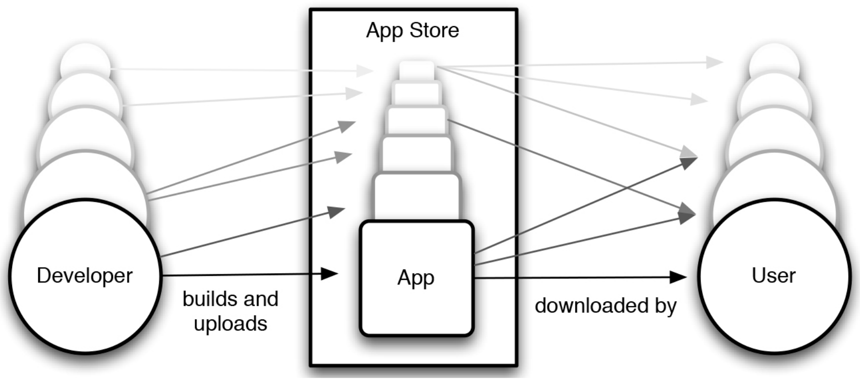


Fig. 1: The interaction between developers, apps, and users in AppEco.

## 2 Background

While the study of mobile app ecosystems is a current and significant topic for researchers, to date there has been little work focussing on the topic [5; 6]. However there is much related work that contextualises and informs our study. One area related to app ecosystems is the study and prediction of app sales and usage. For example, Garg and Telang developed strategies to infer the current sales of an app based on its ranking on Apple’s iOS App Store Top Apps Chart [7]. Such work may enable investors to estimate likely profits should an app reach a specific rank, however there is no certainty that a new app will appear on the chart. Bohmer et al. developed a mobile app to collect mobile app usage information from over 4,100 users of Android devices [8]. Their research revealed interesting app usage behaviours among the users. For example, although users spend almost an hour a day using their phones, an average session with an app lasts less than a minute. They also found that news apps are most popular in the morning and games are at night, but communication apps dominate through most of the day [8]. These studies are informative, but they are limited to studying what is already out there, and “what-if” questions cannot be answered.

In the fields of Alife, Evolutionary Computing and Agent-Based Simulation, researchers have modelled various aspects of ecosystems such as evolutionary dynamics within interacting populations. Classic works in this area include studies by Axelrod and Hamilton on the evolution of cooperation [9] and Maynard Smith and Price on conflicts between animals of the same species [10].

More recently, Holland created Echo, a generic ecosystem model in which evolving agents are situated in a resource-limited environment [11].

Pachepsky et al. investigated the effect of ecological interactions between organisms on the evolutionary dynamics of a community [12]. There are also a growing number of studies on the emergent effects of human interaction at the population level. For example, Kohler et al. used models to understand the environmental and social factors that led to the disappearance of the Puebloan peoples of the North American Southwest [13]. Lux and Marchesi showed that the scaling of financial prices arises from interactions between a large number of market participants [14]. App stores have large populations of apps, developers, and users, and can benefit from similar studies.

Indirect interaction through mechanisms such as stigmergy is commonly studied by Alife researchers. In human society different kinds of objects and tools are often built, adopted, shared and used to support people in their work [15]. It is common for such artefacts to become media for communication (e.g., books, music, and software). One study relating to this topic is the use of robots to create music. In this work, Miranda developed a group of interactive autonomous singing robots that imitate each other to create music [16]. Despite such studies, which often focus on the evolution of human culture with reference to artefacts [17], there is a lack of models that study the development and consumption of artefacts by agents, and how the success of the artefact depends on the preferences of the agents.

## 3 AppEco

In an app ecosystem, coevolving systems of apps, developers, and users form complex relationships, filling niches, competing and cooperating, similar to species in a biological ecosystem [6]. The health of the app ecosystem is largely determined by the communities of developers that create innovative solutions that users want to buy [5; 18]. In an app ecosystem, application software (such as games, medical applications, and productivity tools) that is built for a mobile platform is sold via an app store running on the platform. The app store concept has democratised the software industry – almost anyone can build and sell apps. Once built, an app quickly becomes available to a worldwide market. Mobile device users can download the apps, use them immediately and provide feedback to the developers.

AppEco is an Artificial Life simulation of mobile app ecosystems. The model consists of agents that are abstractions of app users and developers, as well as artefacts that are abstractions of apps. Developer agents build and upload apps to the app store; user agents browse the store and download the apps, see Figure 1. Each download corresponds to a new sale. A distinguishing feature of the AppEco model compared to more traditional agent-based models is the explicit modelling of artefacts as well as the agents that produce and use the artefacts. Different from agents, artefacts are not autonomous, they represent passive entities of the system that are intentionally created and used by agents [15]. App artefacts are important in a model of an app ecosystem because the agents interact with one another via the apps.

## 4 AppEco Components

AppEco consists of app developers, apps, users, and the app store. Each component is described as follows.

### 4.1 Developers

In AppEco, a developer agent represents a solo developer or a team of developers working together to produce an app. Each developer agent has a development duration (*devDuration*, a random value between [$\text{dev}_{min}$, $\text{dev}_{max}$]), which specifies the number of days it needs to build an app. Each developer also records the number of days it has already spent building the app (*daysTaken*).

Each developer is initially active (it continuously builds and upload apps to the app store) but may become inactive (it stops building apps) with probability $P_{Inactive}$. This enables the modelling of part-time developers, hobbyists, and the tendency of developers to stop building apps[2]. Every developer records the number of apps it has already developed and the number of downloads it has received.

Each developer uses one of the following strategies to build apps:

- **S0 Innovator:** Builds an app with random features each time. This strategy models innovative developers. For example, iOS developer Shape Services produces different apps in a variety of categories such as social networking, business, utilities, and productivity[3].
- **S1 Milker:** Makes a variation of own most recent app each time. This strategy models developers who "milk" a single app idea repeatedly. An extreme example is Brighthouse Labs which produced thousands of similar apps[4], such as an app to provide news for each region in each country, and an app about each sports team for each sport.
- **S2 Optimiser:** Makes a variation of own best app each time. This strategy models developers who learn from downloads and improve on their best app. For example, Rovio developed many game apps before hitting the jackpot with Angry Birds. They then built on their success, releasing new apps such as Angry Birds Seasons, and Angry Birds Rio[5].
- **S3 Copycat:** Copies an app in the Top Apps Chart. This strategy models developers who are less creative but want to achieve many downloads quickly. Angry Chickens and Angry Dogs are two example copycats of Angry Birds[6].
- **S* Flexible:** Developers begin with one of the strategies S0-S3. Each developer then has a 0.99 probability to randomly select an app from the Top Apps Chart and change strategy to be the same as the developer of the selected app. There is a 0.01 probability that a strategy is randomly selected.

[2] http://t-machine.org/index.php/2009/06/11/may-2009-survey-of-iphone-developers/
[3] http://www.shapeservices.com/
[4] http://isource.com/2009/05/27/app-store-hall-of-shame-brighthouse-labs/
[5] http://www.wired.co.uk/magazine/archive/2011/04/features/how-rovio-made-angry-birds-a-winner
[6] http://techcrunch.com/2011/12/06/can-we-stop-the-copycat-apps/

These strategies are abstracted after consultation with app literature and developers. Few app developers perform market research before developing.

## 4.2 Apps

Each app artefact is built and uploaded by a developer agent. The features of the app are abstracted as a 10x10 feature grid (**F**) for each app. If a cell in **F** is filled, then the app offers that particular feature. A grid is used so that feature similarity can be represented in the future, e.g., features that are similar can be represented as cells that are near to one another on the grid. For ranking purposes, each app keeps a record of the total number of downloads it has received to date and the number of downloads it has received on each of the previous seven days. For simplicity, the model currently assumes that all apps are sold at the same price; the model of variations in app pricing and categories of apps is left for future work. Each app also records the time when it is being uploaded.

The app feature grid **F** is filled depending on the strategy used by the app's developer:

- **S0:** The cells in **F** are filled probabilistically, such that each cell in the grid has a probability $P_{Feat}$ of being filled.
- **S1:** The cells in **F** are filled probabilistically as in S0 if this is the developer's first app. Otherwise, the developer copies the features of his own latest app with random mutation.
- **S2:** The cells in **F** are filled probabilistically as in S0 if this is the developer's first app. Otherwise, the developer copies the features in his own best app (as determined by the highest daily average downloads) with random mutation. The choice of which app to copy occurs when the developer is starting to build the app. If no apps by this developer have downloads, the developer just copies his most recent app.
- **S3:** An app is randomly selected from the Top Apps Chart and its features are copied with random mutation. The choice of app to copy occurs when the developer is starting to build the app. There is a 0.5 probability that mutation occurs during a copy. Mutation is implemented by randomly selecting a filled cell in **F** and randomly "moving" it to an empty cell in **F**.

## 4.3 Users

Inspired by the recommender systems literature [19], each user agent has preferences (or taste information) that determine the app features that it prefers. The preferences of a user agent are abstracted as a $10\times10$ preference grid (**P**). The cells in **P** are filled probabilistically, such that each cell in the grid has a probability $P_{Pref}$ of being filled. If a cell in **P** is filled, then the user agent desires the feature represented by that cell. If the feature grid **F** of an app has a cell in the same location filled, then it means the app offers a feature desired by the user agent. For example, in Figure 2, all four of the features offered by App 1 match the user agent's preferences, but only two of the features offered by App 2 match the user agent's preferences. For simplicity, preference matching is binary: filled cells either match or do not match. The top right quadrant in **P** is always empty in order to model some features that are undesirable to all users, see Figure 2. For example, no users want an app to have the features of a difficult-to-use or malicious program. Using the AppEco model, a popular app such as Angry Birds can be abstracted as an app with **F** that matches **P** of many users, while a less popular app has **F** that matches few or no users' **P**. The developers are unaware of the users' preferences.

Finally, a user agent keeps a record of the apps it has downloaded, the number of days between each browse of the app store (*daysBtwBrowse*, a random value between [$\text{bro}_{min}$, $\text{bro}_{max}$]), and the number of days that have elapsed since it last browsed the app store (*daysElapsed*). *daysElapsed* is recorded so that the user agent knows when to browse the app store next. When users are initialised, *daysElapsed* is set to be a random number between [0, *daysBtwBrowse*] so that users don't all browse at the same time when they start.

## 4.4 App Store

The app store is the environment used by the agents to store and access apps. Its primary function is to provide a shop front for users and enable them to locate and download apps that match their preferences. To achieve this, it provides three browsing methods: the Top Apps Chart, the New Apps Chart, and Keyword Search. These three methods are modelled because they are common to many app stores, such as iOS, Android, and BlackBerry. The Top Apps Chart ranks apps based on the number of downloads the apps have received. The New Apps Chart displays new apps that have recently been uploaded by developer agents; only a small subset of new apps is chosen for this chart.

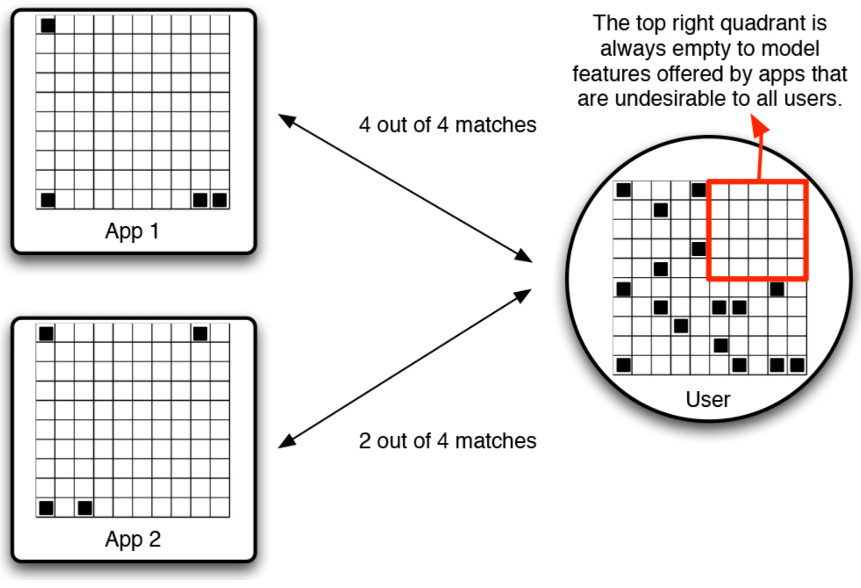

The top right quadrant is always empty to model features offered by apps that are undesirable to all users.
4 out of 4 matches
2 out of 4 matches
App 1
App 2
User

Fig. 2: Matching app features with user preferences.

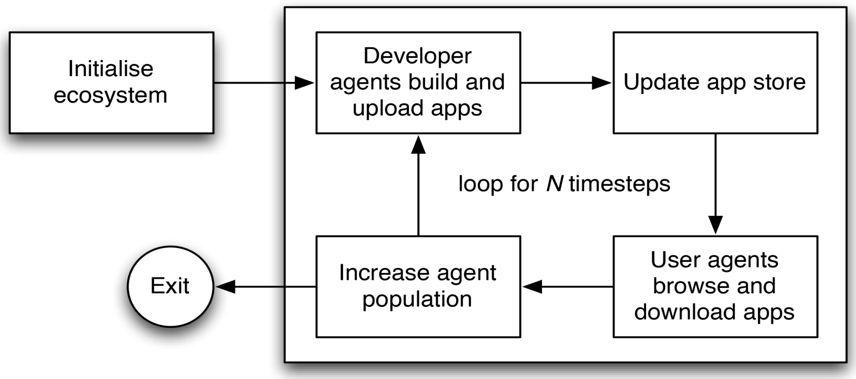

Initialise ecosystem
Developer agents build and upload apps
Update app store
loop for N timesteps
User agents browse and download apps
Increase agent population
Exit

Fig. 3: The AppEco algorithm.

Keyword Search returns a list of apps that match the keyword entered by the user agent. In AppEco, Keyword Search is abstracted as a random search for a random number of apps. It is implemented in this way because keywords may not correspond to features, so a matching keyword does not mean the app has desirable features for the user.

## 5 AppEco Algorithm

The AppEco algorithm models the daily interactions between the AppEco components described in the previous section. Each timestep in the algorithm represents a day in the real ecosystem.

Inspired by the ecology literature [20; 21], the population growth of user and developer agents is modelled using a sigmoid growth function commonly used to model the population growth in natural systems. The equation models the growth rate of user and developer agents in an app ecosystem declining as their population density increases, with the size of the ecosystem limited by the market share of the mobile platform. The population size at timestep $t$, $pop_t$, is defined by,

$$pop_t = MinPop + \frac{(MaxPop - MinPop)}{1 + e^{S*t-D}} \tag{1}$$

where MinPop is the minimum population, MaxPop is the maximum population, S determines the slope of the growth curve (S is negative for a growth curve), and D shifts the curve from left to right. Different growth formulas can be used to model different ecosystems [20; 21].

The AppEco algorithm, see Figure 3, is detailed as follows.

**Initialise ecosystem.** This step launches AppEco with the population of developer and user agents as defined in Eq. 1, with timestep $t = 0$. It is common for app stores to have apps before it is opened. For example, the iOS App Store had 500 apps the day it was launched[7]. As such, this step also creates an initial number of app artefacts (N$_{InitApp}$). The developers of these initial apps are randomly selected from the pool of initial developers of strategy S0, S1 or S2 (S3 waits for apps to be on Top Apps Chart before it builds apps). The attributes of initial developers, apps, and users are set as described in the previous section.

[7] http://www.apple.com/pr/library/2008/07/10iPhone-3G-on-Sale-Tomorrow.html

**Developer agents build and upload apps.** For each active developer, *daysTaken* is incremented by 1. If *daysTaken* exceeds this developer's *devDuration*, the app is completed. The developer then uploads the app to the store, resets *daysTaken* to 0, and decides on the next app to build. The feature grid **F** of the app is set depending on the developer's strategy.

**Update app store.** The New Apps Chart is updated. When timestep $t = 0$, the New Apps Chart consists of a random selection of initial apps. In each following timestep, each new app has a probability $P_{OnNewChart}$ of appearing on the New Apps Chart. Apps are randomly selected here because the selection criteria are not the focus of this work and real app stores do not reveal how they select apps for the New Apps Chart. The maximum number of apps in the chart is defined by $N_{MaxNewChart}$. As newly selected apps are added to the chart, older apps appear lower in the chart and are no longer listed when their position exceeds the chart size. The Top Apps Chart is also updated. When timestep $t = 0$, the Top Apps Chart is empty because no apps have been downloaded yet. In each following timestep, apps are ranked in the order of decreasing score, calculated as $8*D_1+5*D_2+5*D_3+3*D_4$ where $D_n$ is the number of downloads received by the app on the nth day before the current day [22]. The maximum number of apps in the Top Apps Chart is defined by $N_{MaxTopChart}$.

**User agents browse and download apps.** For each user, *daysElapsed* is incremented by 1. If *daysElapsed* exceeds *daysBtwBrowse*, then the user browses the app store, and resets *daysElapsed* to 0. The user browses the New Apps Chart and the Top Apps Chart, and conducts Keyword Search (which returns a random number of apps between [$key_{min}$, $key_{max}$]). The user browses each app that it has not previously downloaded: the feature grid of the app is compared with the preference grid of the user. If all the features offered by the app match the user's preferences, then the user downloads the app. For example, in Figure 2, the user downloads App 1 but not App 2.

**Increase agent population.** This step increases the number of user and developer agents in the ecosystem for the next timestep, using Eq. 1.

AppEco is implemented in C++ and the code can be requested from the authors via email. It is developed to be highly configurable so that it can simulate various app ecosystems, such as iOS, Android, and BlackBerry.

## 6 Experiments

In order to investigate the effects of developer strategies in AppEco, we must first calibrate the simulation to match, as much as is feasible, the behaviour of a real app store. We select Apple's iOS App Store for our experiments, as it is one of the oldest and most established app stores. The calibration of AppEco to the iOS App Store is described in Section 6.1. We then investigate how different developer strategies affect individual and collective success in AppEco. Two experiments are conducted. Experiment 1 (E1) in Section 6.2 investigates the success of each individual strategy S0, S1, S2, and S3. Experiment 2 (E2) in Section 6.3 investigates the more realistic scenario of competing strategies by having developers select their own strategies over time. Thus in E2, developers have strategy S*, and the overall success of the App Store is compared against E1.

### 6.1 Calibrating AppEco for iOS

We collected the following iOS data over a period of three years, from the start of the iOS ecosystem in July 2008 (Q4 2008) until the end of June 2011 (Q3 2011):

- **Number of iOS developers.** The number of iOS developers is based on the number of worldwide iOS developers month over month compiled by Gigaom[8].
- **Number of iOS apps and downloads.** The number of apps and downloads is based on statistics provided in Apple press releases and Apple Events[9]. For example, in the Apple Special Event on 9th September 2009, Apple CEO Steve Jobs announced the App Store to reach 75,000 apps and 1.8 billion downloads, and Apple's press release on 28th September 2009 announced that the App Store has achieved more than 85,000 apps and 2 billion downloads[10].

[8] http://gigaom.com/apple/infographic-apple-app-stores-march-to-500000-apps/
[9] http://www.apple.com/apple-events/
[10] http://www.apple.com/pr/library/

| | | | |
|---|---|---|---|
| [Pop$_{minUser}$, Pop$_{maxUser}$] | [1500, 40000] | [dev$_{min}$, dev$_{max}$] | [1, 180] |
| D$_{User}$ | -4.0 | P$_{Pref}$ | 0.45 |
| S$_{User}$ | -0.0038 | P$_{Feat}$ | 0.04 |
| [Pop$_{minDev}$, Pop$_{maxDev}$] | [1000, 120000] | P$_{OnNewChart}$ | 0.001 |
| D$_{Dev}$ | -4.0 | N$_{MaxNewChart}$ | 40 |
| S$_{Dev}$ | -0.005 | N$_{MaxTopChart}$ | 50 |
| N$_{InitApp}$ | 500 | P$_{Inactive}$ | 0.0027 |
| [bro$_{min}$, bro$_{max}$] | [1, 360] | [key$_{min}$, key$_{max}$] | [0, 50] |

Tab. 1: Constant Values Resulting from iOS Calibration.

- **Number of iOS users.** The number of iOS users is based on the number of iOS devices (iPod Touch, iPhone, and iPad) sold by Apple over time. The sales figures are available from Apple's quarterly financial data[11], and for simplicity the calculation assumes that each user has one iOS device.

We calibrated AppEco to simulate the iOS app ecosystem. Table 1 summarises the calibrated values for the system constants. Most constants were set from publicly available data. For example, the total number of people in the world who use mobile devices is approximately 4 billion [23]. According to the International Data Corporation (IDC), Apple had 2.8% of the mobile device market share in Q1 2010 and 5% in Q1 2011[12]. By assuming a maximum increase of market share to be 10%, our calculation gives us a Pop$_{maxUser}$ of 400 million users. In order to match (curve-fit) the iOS user and developer growth rates, values such as D and S for users and developers were determined through tuning experiments. To ensure that the system is computationally feasible, one app represents one real app, and one developer agent represents one real developer, but one user agent represents 10000 real users. This is because it is computationally infeasible in terms of memory to simulate hundreds of millions of users.

Figure 4 illustrates the actual and simulated number of users, developers, apps, and downloads. As can be seen, the behaviour of AppEco closely resembles the behaviour of the iOS ecosystem, including emergent rates such as the number of apps and downloads.

[11] http://www.apple.com/pr/library/
[12] http://www.idc.com/getdoc.jsp?containerId=prUS22808211

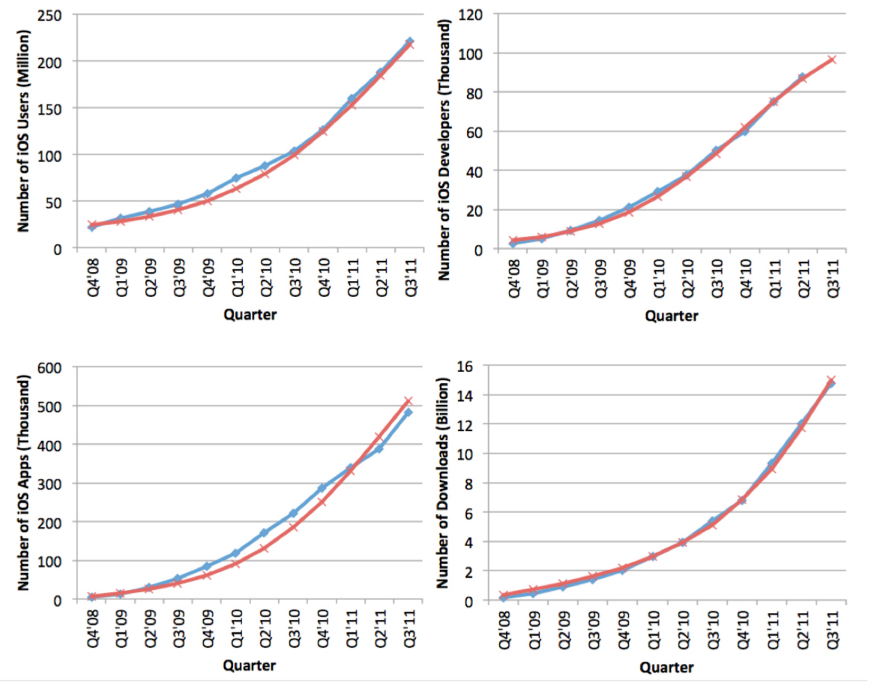


Fig. 4: Actual vs. simulated number of iOS users, developers, apps, and downloads (blue is actual, red is simulated).

A run of the simulation takes approximately 16 seconds CPU time on a MacBook Air with a 1.8GHz Intel Core i7 Processor and 4GB of 1333 MHz DDR3 memory. After three years (1080 timesteps assuming 30 days a month), the model typically contains more than 100,000 developer agents, 500,000 apps, 20,000 user agents (corresponding to 200m real users), and 1.5 million downloads (corresponding to 15bn real downloads).

## 6.2 E1: Comparing Strategies

In a mobile app ecosystem, developers compete with each other to earn more downloads. Some developers try many different ideas, some produce many similar apps, some gain experience from their previous successful apps, and some copy successful apps created by other developers. To learn how each strategy performs relative to one another, we ask the following research questions:

- **RQ1:** Which developer strategy enables individual developers to be most successful?
- **RQ2:** What is the diversity of apps produced by each strategy?
- **RQ3:** Which developer strategy enables the developer to become more successful as they develop more apps?

To answer RQ1, the following measurements are used.

**Average Downloads per App (*AvgDl*):** For each strategy S0 to S3, *AvgDl* is the total number of downloads received by the developers of the strategy, divided by total number of apps built by the developers of the same strategy.

**Top 20 Total Downloads (*Top20TotDl*):** The developers of all strategies are ranked based on the total number of downloads they received. For each strategy S0 to S3, *Top20TotDl* is the proportion of developers in the top 20 of this list that belong to the current strategy.

**Top 20 Average Downloads (*Top20AvgDl*):** The developers are ranked based on the average number of downloads they received per app. For each strategy S0 to S3, *Top20AvgDl* is the proportion of developers in the top 20 of this list that belong to the current strategy. Developers can receive more total downloads by building more apps – this measure identifies efficient developers who have many downloads with few apps.

**Zero Downloads (*ZeroDl*):** For each strategy S0 to S3, *ZeroDl* is the proportion of developers that belong to the current strategy who have received no downloads for any of their apps so far.

To answer RQ2, the Feature Coefficient of Variation (*FeatCV*) is used to measure the app coverage of features that are desired by users. For each cell in the desired region of feature grid **F**, we calculate the number of apps that offer that feature, forming a combined feature grid $\mathbf{F}^C$. *FeatCV* is the coefficient of variation of grid $\mathbf{F}^C$. *FeatCV* is defined in Eq. 2 and expressed as a percentage.

$$FeatCV = \sigma/\mu \tag{2}$$

where $\sigma$ is the standard deviation and $\mu$ is the mean of values in grid $\mathbf{F}^C$. In this simulation, the user preference coefficient of variation in the desired region of **F** is 0.68%, indicating that the mean preferences for the user population are evenly distributed over the mean $\mathbf{F}^C$ feature grid. As such, a good strategy should have a low *FeatCV*, which means that all the apps have features that cover the desired region in **F** evenly (in combination they better meet all the users' needs).

To answer RQ3, we measure the *Fitness* of each strategy as its developers gain more experience in app development. For each strategy, we categorised the apps into classes corresponding to their developers' first apps, second apps, third apps, and so on. These correspond to the apps created by the developers at experience level 1, 2, 3, and so on. For each app, we "survey" the users and ask if they would download the app: if all the features in the app match the user's preferences then they would download the app. For each strategy, the *Fitness* of the strategy at experience level $L$ is defined in Eq. 3.

$$Fitness_L = \frac{AvgDl_L}{NumUsers} \tag{3}$$

where $AvgDl_L$ is the number of potential downloads as reported by users in the survey for all the apps in experience level $L$ divided by the number of apps in $L$, and $NumUsers$ is the number of users who participated in the survey. $Fitness_L$ ranges from 0 to 1. The higher the value, the fitter the strategy.

AppEco was run with the settings described in Section 6.1. Throughout each run, developers in the ecosystem were randomly assigned strategies S0, S1, S2 or S3 in equal proportions to enable direct comparison of relative performance. AppEco was run for 1080 timesteps (corresponding to three years in the real world, assuming 30 days a month). The experiment was repeated 100 times. The results were averaged over the 100 runs.

**RQ1: Which developer strategy enables individual developers to be most successful?** As can be seen in Table 2, the Copycat strategy S3 is the most successful, receiving the highest *AvgDl*, *Top20TotDl* and *Top20AvgDl*, and the lowest *ZeroDl*. (The success of Copycats is well known in the real world – several Copycats who have parasitised Angry Birds have risen high in the Top Apps Chart[13].) Although the Innovator strategy S0 performed the worst with the lowest *AvgDl* and *Top20TotDl*, it has a lower number of developers with zero downloads (*ZeroDl*) compared to strategies S1 and S2. This is because by randomly trying different ideas for apps, creative S0 avoids dwelling on ideas that do not work, unlike S1 Milker and S2 Optimiser that keep working on similar apps.

[13] http://techcrunch.com/2011/12/06/can-we-stop-the-copycat-apps/

S1 Milker has the lowest *Top20AvgDl*, illustrating that a strategy that produces many similar apps results in poor performance. (In the real App Store this strategy is also heavily criticised by other developers and users for its exploitative approach[14].)

Studies of individual runs show that although S3 Copycat dominates the top developers lists, individual developers from other strategies can become the most successful developer in the ecosystem, achieving either the highest number of downloads or the highest average downloads per app. However, the occurrence is by chance: the developer has to build the right app at the right time (e.g., the app has features preferred by many users, appears on the New Apps Chart, is downloaded by many users, appears on the Top Apps Chart, and continues to attract downloads). Developers with high *Top20AvgDl* tend to develop one app in the entire three years. This shows that it is difficult to repeat success (a fact well-known to app developers in the real world[15]). The more apps a developer builds, the lower the average downloads tends to be. As such, the same developers rarely appear on both average and top downloads lists. In many runs, we observe that developers who built many apps can have the highest total number of downloads, but often have a low average.

**RQ2: What is the diversity of apps produced by each strategy?** Developers who use the S0 Innovator strategy offer the most even coverage of features. Despite being the most successful strategy in terms of downloads, S3 Copycat has the highest *FeatCV*, which suggests that it only partially covers the preference space of users. Analysis shows that S3 produces an average of about 1 feature per app, while the other strategies have an average of 4 features per app.

This is because users only download an app when all the features of the app matches their preference, and as a result apps with fewer features will have a higher chance to be downloaded by many users and appear on the Top Apps Chart. Since S3 Copycat copies from Top Apps Chart, the Copycat developers are likely to copy apps with very few features. (This is consistent with the advice from successful app developers – apps with fewer features have a higher chance of being downloaded[16].)

[14] http://isource.com/2009/05/27/app-store-hall-of-shame-brighthouse-labs/
[15] http://www.fastcompany.com/1792313/striking-it-rich-in-the-app-store-for-developers-its-more-casino-than-gold-mine
[16] http://www.fastcompany.com/1792313/striking-it-rich-in-the-app-store-for-developers-its-more-casino-than-gold-mine

To investigate further, we group the apps by the experience level of their developers when the apps are built. We aggregate the features of the apps with the same strategy and experience level. We find that the coverage of features using S3 Copycat is worse than other strategies indicating that developers following this strategy are not satisfying as many users' needs (Figure 5). In contrast, the developers using S2 Optimiser correctly avoid the top right corner as they become more experienced, while also consistently covering the features desired by the users.

**RQ3: Which developer strategy enables the developer to become more successful as they develop more apps?** The only strategy that shows clear improvement as the developers become more experienced is S2 Optimiser. While S3 Copycat is the clear winner in terms of downloads, developers using the strategy are plagiarising the work of other developers rather than improving their own work. In S2 developers release new apps based on mutated copies of their most successful apps. As such this is similar to a $(1 + \lambda)$ Evolutionary Strategy. Figure 6 illustrates that among the four strategies, S2 shows a classic evolutionary curve. This demonstrates that developers who improve their own apps based on download feedback should increasingly meet the needs of the users. In addition, among all four strategies, S2 also offers the highest number of features desired by the users.

## 6.3 E2: Ecosystem Health

In real life, there are no fixed strategies. Developers can choose the strategy they want to use. With all developers free to choose, the strategies directly compete with each other in the ecosystem. If a strategy were more effective then it might quickly dominate all others; less effective strategies might become a tiny minority. However, in many ecosystems, individual success is not always reproducible for many [9]. It is therefore of great interest to study how the number of developers using each strategy varies over time. Our research questions are thus:

- **RQ4:** When strategies compete, how often is each strategy chosen by developers?
- **RQ5:** What is the diversity of apps produced?
- **RQ6:** Is an app ecosystem that comprises competing strategies able to improve its performance in the long term?

| | AvgDl | Top20TotDl | Top20AvgDl | ZeroDl | FeatCV |
|---|---|---|---|---|---|
| S0 Innovator | 1.18 (0.14) | 3.85% (4.20%) | 9.10% (5.52%) | 26.51% (0.23%) | **1.40% (0.13%)** |
| S1 Milker | 1.19 (0.14) | 5.95% (5.16%) | 8.80% (6.52%) | 32.85% (0.26%) | 4.27% (0.35%) |
| S2 Optimiser | 1.41 (0.15) | 7.40% (6.05%) | 9.25% (7.30%) | 32.90% (0.27%) | 6.50% (0.62%) |
| S3 Copycat | **8.22 (0.41)** | **82.80% (8.30%)** | **72.85% (10.03%)** | **7.74% (0.23%)** | 54.36% (6.78%) |

Tab. 2: Results for RQ1 and RQ2 (Std. Dev. in brackets).

| | Proportion of Developers | Standard Deviation |
|---|---|---|
| S0 Innovator | **33.51%** | 19.74% |
| S1 Milker | 26.57% | 17.17% |
| S2 Optimiser | 29.26% | 18.94% |
| S3 Copycat | 10.67% | 7.65% |

Tab. 3: Proportion of Developers at timestep $t = 1080$ (RQ4).

To answer RQ4, we measure the proportion of developers using each strategy over time. To answer RQ5, we use *FeatCV* (Eq. 2) on the aggregated app features in E2 and compare the results with the aggregated app features in E1. To answer RQ6, we use the *Fitness* measure (Eq. 3) on E2 and compare the outcome with E1. AppEco was run with the settings described in Section 6.1, with all developers initialised with Flexible strategy S*. E2 was also run for 1080 timesteps and repeated 100 times. The results were averaged over the 100 runs.

**RQ4: When strategies compete, how often is each strategy chosen by developers?** The choice of strategy depends on the proportion of other strategies in the population, but S3 Copycat is the least frequently chosen strategy (Table 3), despite appearing to be the best strategy from E1. When developers have a choice, very quickly the Copycat strategy is dropped in favour of the other strategies. As is evident from Table 3, strategy S0 Innovator is the most popular choice, followed by S2 Optimiser and then S1 Milker. However, the standard deviations for S0, S1 and S2 are very high, indicating that the percentage of developers from those strategies differs greatly in different runs (Table 3).

Indeed, studies from individual runs reveal that while S3 Copycat consistently falls quickly to 10% of the developer population, the winning strategies fluctuate unpredictably, at times with S0 Innovator, S1 Milker or S2 Optimiser each dominating the population, see Figure 7. It is interesting to note that the two most widely hated strategies in real life: S1 Milker and S3 Copycat, appear to be used the least in the ecosystem, with S3 Copycat clearly in the minority. To assess whether S3 could ever become widely adopted, we repeated E2 by only allowing developers to change strategies after 2 months in order to give S3 more opportunity to work. There was no change to the result: S3 again becomes unpopular. Even when E2 is repeated with 50% developers using S3 Copycat, most developers subsequently avoid choosing S3.

In fact, in E2, the only way to guarantee that S3 will dominate the ecosystem is to force developers not to change strategies until after 1 year, by which time there are plenty of good apps to copy.

This demonstrates that S3 is only viable as a minority strategy in a healthy ecosystem. Copycats rely on good apps created by other strategies; it is extremely difficult for an ecosystem to support a large proportion of Copycats. The result mirrors the app stores in the real world — Copycat developers regularly appear and take advantage of the success of others, but nevertheless their strategy remains in the minority.

**RQ5: What is the diversity of apps produced?** When developers can choose their strategy, the app ecosystem has a higher diversity of apps. E1 resulted in a *FeatCV* of 6.28% with a standard deviation of 0.72%; E2 resulted in *FeatCV* of 2.87% with a standard deviation of 2.33%.

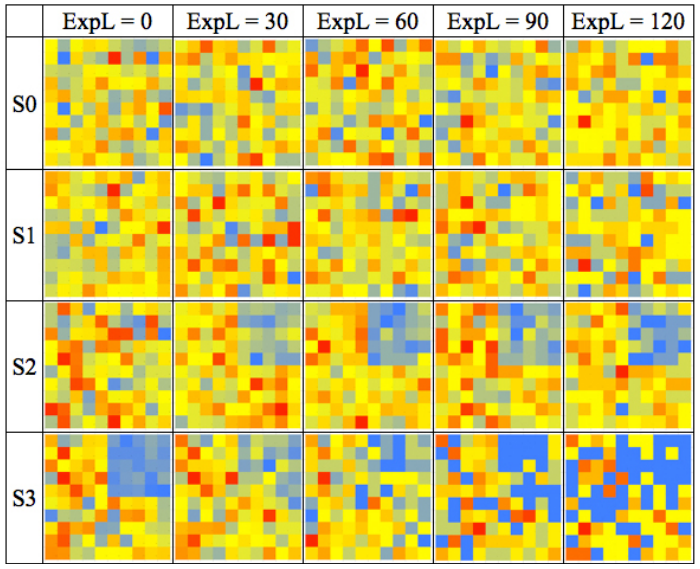


Fig. 5: Heat maps for one run showing total app features over experience level.

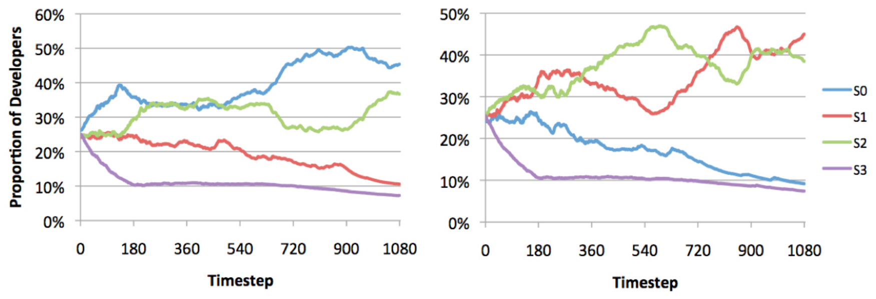


Fig. 7: Proportion of developers in two example runs.

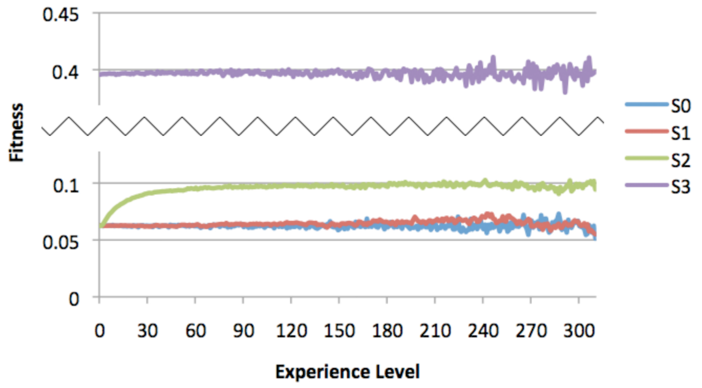


Fig. 6: Fitness of a strategy as its developers become more experienced. Later data is more sparse as fewer developers create large number of apps, resulting in more noise.

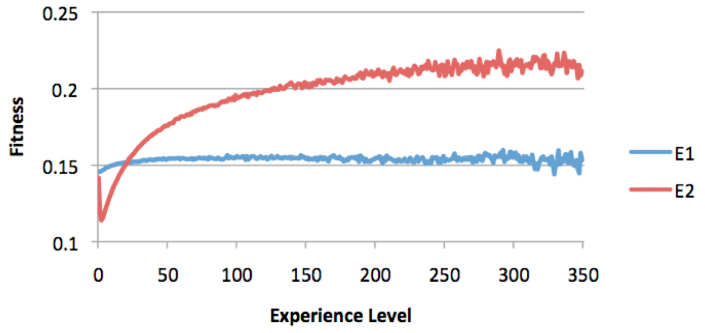


Fig. 8: Fitness as developers become more experienced.

This shows that in E2, the apps evenly cover the users' preference space. Analysis of all features for E1 and E2 shows that E2 has a more evenly distributed feature set. This means that in E1, users may find that the app store does not have any apps that meet some of their preferences.

**RQ6: Is an app ecosystem that comprises competing strategies able to improve its performance in the long term?** When we plot total fitness against developer experience level, there is a clear improvement evident for the more realistic ecosystem where developers are free to choose their strategies. Figure 8 illustrates that the developers in E2 improve more as they become more experienced, indicating that flexible developers who are not all locked into one development strategy will collectively perform better as they develop more apps.

## 7 Conclusions

It is the dream of many developers to make their fortune with a clever app. But when the chances of success are now similar to the chances of winning the lottery – and falling daily as new apps are released – how can anyone be a successful app developer? In this work, we presented AppEco, an Artificial Life agent-based model that simulates app ecosystems, and investigated these issues. AppEco models developers (agents that build apps) and users (agents that download apps). It simulates the app store environment and the population growth of the agents and apps. Significantly, AppEco also models apps (artefacts produced by the developers and downloaded by users) and their features. AppEco is a complex ecosystem where developers, users and apps increase continuously, and interaction strategies may continuously change. Our experiments investigated different developer strategies: Innovators, Milkers, Optimisers, and Copycats.

In a complex ecosystem no strategy can be a guaranteed winner, but our results indicate that some strategies should be chosen more frequently than others. Innovators produce diverse apps, but they are hit or miss – some apps will be popular, some will not. Milkers may dwell on average or bad apps as they churn out new variations of the same idea. Optimisers produce diverse apps and tailor their development towards users' needs. Finally, Copycats may seem like the best strategy to guarantee downloads in an app ecosystem, but the strategy can only work when there are enough other strategies to copy from.

In addition, this strategy can only exist in a minority, otherwise app diversity will decrease (many duplicated apps result in a scarcity of some features desired by users) and the fitness of the ecosystem will suffer. This study is one of many we will be undertaking with AppEco. We plan to study the effect of publicity on app downloads, and understand how a user might best locate desirable apps and communicate their requirements and feedback about the apps to developers, and how these feedback can influence app development. AppEco can also be calibrated to study other app ecosystems, such as Android and Blackberry, and extended to model web-based platforms such as Facebook and Chrome.

## References

[1] Chen, B. X. 2009. Coder's Half-Million-Dollar Baby Proves iPhone Gold Rush Is Still On. Wired. (12 Feb).

[2] Dredge, S. 2012. Angry Birds bags 6.5m Christmas Day downloads. *The Guardian*. (4 Jan).

[3] Baghdassarian, S., and Milanesi, C. 2010. *Forecast: Mobile Application Stores, Worldwide, 2008-2014*. Gartner.

[4] Distimo. 2011. *In-depth view on download volumes in the Google Android Market*. Utrecht, Netherlands.

[5] Jansen, S., Finkelstein, A., and Brinkkemper, S. 2009. A sense of community: a research agenda for software ecosystems. *In Proc. of 31st Int. Conf. on Software Engineering (ICSE) - Companion Volume*, p. 187-190.

[6] Lin, F., and Ye, W. 2009. Operating System Battle in the Ecosystem of Smartphone Industry. In *Int. Symp. on Information Engineering and E-Commerce*, p. 617-621.

[7] Garg, R., and Telang, R. 2011. *Estimating App Demand from Publicly Available Data*. School of Information Systems and Management, Heinz College, Carnegie Mellon University.

[8] Bohmer, M., Hecht, B., Schoning, J., Kruger, A., and Bauer, G. 2011. Falling asleep with Angry Birds, Facebook and Kindle: a large scale study on mobile application usage. In *MobileHCI 2011*, p. 47-56.

[9] Axelrod, R., and Hamilton, W. D. 1981. The evolution of cooperation. *Science*. 211(4489): p. 1390.

[10] Smith, J. M., and Price, G. 1973. The Logic of Animal Conflict. *Nature*. 24615.

[11] Holland, J. H. 1992. *Adaptation in natural and artificial systems*. MIT Press, Cambridge, MA.

[12] Pachepsky, E., Taylor, T., and Jones, S. 2002. Mutualism promotes diversity and stability in a simple artificial ecosystem. *Artificial Life*. 8(1): p. 5-24.

[13] Kohler, T. A., Gumerman, G. J., and Reynolds, R. G. 2005. Simulating ancient societies. *Scientific American*. 293(1): p. 76-84.

[14] Lux, T., and Marchesi, M. 1999. Scaling and criticality in a stochastic multi-agent model of a financial market. *Nature*. 397(6719): p. 498-500.

[15] Omicini, A., Ricci, A., and Viroli, M. 2008. Artifacts in the A& A meta-model for multi-agent systems. *Autonomous Agents and Multi-Agent Systems.* 17(3): p. 432-456.

[16] Miranda, E. R. 2008. Emergent songs by social robots. *Journal of Experimental & Theoretical Artificial Intelligence*. 20(4): p. 319-334.

[17] Wheeler, M., Ziman, J., and Boden, M. A. 2002. *The Evolution of Cultural Entities*. OUP/British Academy.

[18] Cusumano, M. A. 2010. Platforms and services: Understanding the resurgence of Apple. *Communications of the ACM*. 53(10): p. 22-24.

[19] Adomavicius, G., and Tuzhilin, A. 2005. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Trans. on Knowledge and Data Engineering*. 17(6): p. 734-749.

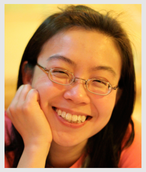
[20] Kingsland, S. E. 1995. *Modeling nature: episodes in the history of population ecology*. University of Chicago Press.

[21] Gershenfeld, N. A. 1999. *The nature of mathematical modeling*. Cambridge University Press.

[22] Lanfranchi, M.-C., Benezet, B., and Perrier, R. 2010. *How to Successfully Market your iPhone Application.* faberNovel.

[23] International Telecommunication Union. 2009. *Yearbook of Statistics - Chronological Time Series 2000-2009.* p. 302.

## About the authors



Dr. Soo Ling Lim is a Research Associate at the UCL Interaction Centre in University College London, an Assistant Professor at the Honiden Laboratory, National Institute of Informatics in Japan, a Visiting Professor at the Technical University of Loja in Ecuador, and a Research Associate at the University of Bournemouth in UK. Soo Ling received a Ph.D. in large-scale software requirements engineering from the University of New South Wales, Australia in 2011. Before her PhD, she was an ERP analyst programmer and a SAP consultant with the Computer Sciences Corporation. She was also a software engineer at CIC Secure, an Australian-based company specialising in electronic key management and asset security systems. Soo Ling's research investigates mobile app ecosystems, social networks, and requirements elicitation techniques for large software projects.

Homepage: http://soolinglim.wordpress.com/
Email: s.lim@cs.ucl.ac.uk

Dr. Peter J. Bentley is an Honorary Reader at the Department of Computer Science, University College London (UCL), Collaborating Professor at the Korean Advanced Institute for Science and Technology (KAIST), Visting Fellow at SIMTech, A*STAR, Singapore, a contributing editor for WIRED UK, a consultant and a freelance writer. Peter runs the Digital Biology Interest Group at UCL. His research investigates evolutionary algorithms, computational development, artificial immune systems, swarming systems and other complex systems, applied to diverse applications including design, control, novel robotics, nanotechnology, fraud detection, mobile wireless devices, security, art and music composition. He is also author of the number one bestselling iPhone app iStethoscope Pro. He has published over 200 scientific papers and is author of eight books including the recent "Digitized".

Homepage: www.peterjbentley.com/
Email: p.bentley@cs.ucl.ac.uk

# Distilling GeneChips with GP on the Emerald GPU Supercomputer

William B. Langdon, CREST Centre, Department of Computer Science,
University College London, W.Langdon@cs.ucl.ac.uk

The Emerald supercomputer contains 1008 x86 CPU cores and 372 nVidia M2090 Tesla. A CUDA GPGPU genetic programming GeneChip datamining application which searches for non-linear gene expression based prediction of long term survival following breast cancer surgery was transferred without change and run on part of the Emerald cluster. An average of 33 giga GPopS$^{-1}$ was achieved.

## 1 Introduction

The Emerald super computer has recently been installed. It contains 372 top-end nVidia Tesla GPU processors (see Figure 1). I will report performance of an existing genetic programming breast cancer long term outcome prediction data mining application [5] on Emerald.

Emerald is shared by the UK Science and Technology Facilities Council (STFC) and four UK universities (Oxford, Bristol, Southampton and UCL). The next section summarises the existing GPGPU ten year tumor application. Then Section 3 shares some initial experiences of Emerald. This is followed by world beating performance results from it (Sections 4). In Section 5, I discuss problems with file access and allocating GPU Tesla to jobs.

## 2 Predicting Breast Cancer Long Term Survival

For three years (1987–1989) samples were taken from most of the women who underwent surgery for breast tumours in Uppsala in southern Sweden. The biopsies were subsequently measured using Affymetrix GeneChips [9] generating more than a million data points for each of the 251 patients. We obtain these gene expression data via NCBI's GEO, checked them for spatial errors, quantile normalised them [6; 2] and then used genetic programming [11] to datamine them eventually yielding a small predictive model [5]. (The normalised data are now available via http://groups.csail.mit.edu/EVO-DesignOpt/GP Benchmarks/uploads/Main/GSE3494/).

The original genetic programming work used an nVidia GeForce 8800 GTX graphics processing unit (GPU) (with 128 stream processors) and RapidMind C++ software. (In total Emerald has 190 464 stream processors.) The RapidMind software was recently rewritten in nVidia's CUDA and the original experiments re-run on a C2050 Tesla GPU donated by nVidia [3] (code available via FTP). The C2050 code has been run without modification on Emerald.

The genetic programming approach [4] is somewhat unusual in that instead of trying to solve the datamining problem in one go it uses several phases in which multiple independent GP runs are used to select which of the gene expression variables convey enough information to be useful in evolving a final non-linear predictive model of breast cancer survival. (The models are quite small, on average they contain only 12.9 components.)

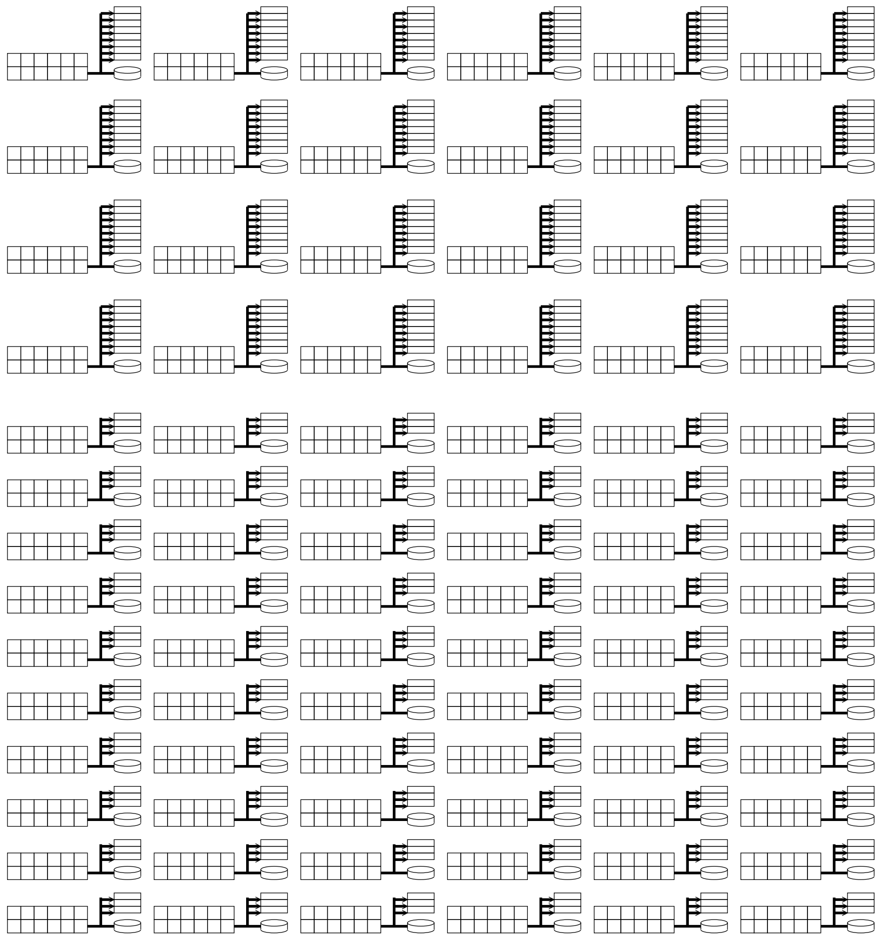


Fig. 1: Schematic of Emerald GPU supercomputer. Each node consists of 6 twin-core CPU (squares), local disk and 3 or 8 nVidia Tesla M2090 GPUs each containing 512 stream processors. All 84 cluster nodes are connected by QDR Infiniband with Mellanox switches in a fat-tree topology to a 135000 GBytes Panasas storage system.

Our goal was simply to demonstrate re-running this approach on Emerald. Other experiments might use different numbers of phases and different numbers of independent runs in each phase.

For the breast cancer data there are three phases (see Figure 2). The first two phases each consist of 100 independent runs, each with a population of 5 million non-linear breast cancer predictors evolved from generation zero to generation ten using the data from 91 women for fitness training to select good gene expression models. In each GP run all 5.2 billion fitness tests are done on the GPU (taking a total of about ten seconds). The other operations remain on the host (although these too might in future be run on the GPU). Calculating fitness (before GPUs) used to totally dominate run time. Now operations which used to be a trivial part of the total time are significant and often the host operations take four or five times as long as those on the GPU.

At the end of each run in the first two phases, 8 000 good non-linear models are harvested and the gene expression data they use is extracted. Only gene expression data from good models is passed to the next phase. There is a single run in the third phase. It takes the eight best of the original 1 013 888 variables filtered by the first and then second phases and generates the final predictive model.

## 3 Problems

The Emerald NFS disk system will happily cache even large files (like the genetic programming training data, 352 Megabytes) but does not broadcast data. So, if twenty nodes try to read the same file, NFS simultaneously provides 20 copies of it. Even so it takes on average only 1.6 seconds to read the 352 megabytes of training data (A combined data rate of about 30 billion bits/second.) The training data is saved by `rsync` on each node's local disk (`/tmp`) so that if the node is used again, e.g. by the second pass, the training does not have to be read again. Also separating transferring the training data across Emerald's network from the CUDA code can make it easier to diagnose problems. For larger files or in cases where file I/O is more critical it could make sense for a single node to read the file and then broadcast it to the others, e.g. via MPI or hdf5.

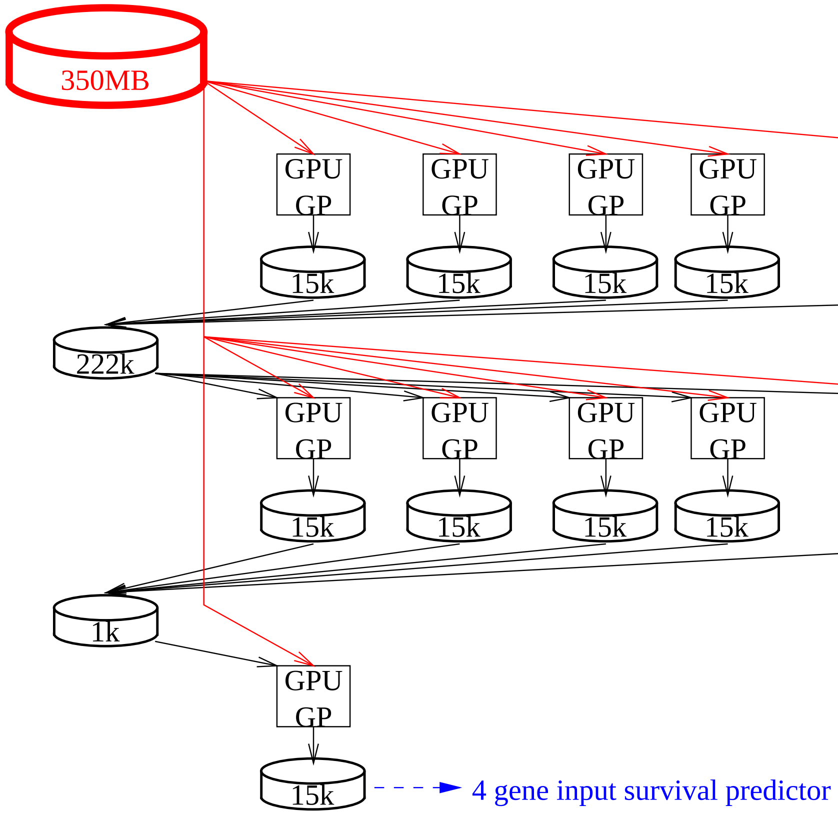


Fig. 2: Schematic of major data flows when datamining breast cancer dataset. (GeneChip training data in red.) Only 5 of the 100 GPU GP runs in the first two phases are shown. The last phase (bottom) consists of a single GPU GP run which generates the final simple model which uses only four of the millions of GeneChip data to predict long term survival following breast tumour surgery.

The LSF batch queue software was written to share CPU cores but as yet the Emerald LSF does not know about GPUs. Finding out which GPUs are in use is tricky. The simplest approach is to use the LSF `-x` switch to ensure each job has exclusive access to the nodes it runs on (and therefore exclusive access to their GPUs). Having said that, there is an unresolved problem whereby occasionally jobs fail with CUDA claiming a GPU is already in use (`error 46`) when there aren't any other LSF jobs using it. LSF `-x` is also wasteful since each of Emerald's nodes has twelve cores but many of our jobs will only use three (or possibly eight).

There are a few little things which I found useful which are recorded in this web page http://www.cs.ucl.ac.uk/staff/W.Langdon/emerald/ and technical report [7].

## 4 Results

The LSF cluster queue management system can organise the multiple GP runs and various passes in many different ways. The final configuration (see Figure 3) was chosen to try and extract the maximum amount of parallelism by running all 100 GP runs in each pass in parallel but also to minimise the number of times the training data has to be copied. The GP runs are pretty uniform. If running well they each take approximately the same time. Therefore synchronising all one hundred should not ideally lead to much processing resources being wasted. Figure 3 distinguishes between nodes with three GPU and those with eight. LSF is asked to allocate 12 (of the 60) three GPU nodes and 8 (of the 24) eight GPU nodes. This is done as two LSF job arrays (one with 12 elements and the second has 8). In both arrays each job exclusively uses the whole of the Emerald node it is allocated (including the GPUs). This takes a total of 20 nodes.

With this LSF configuration the application needs 20 copies of the training data. Since in this case LSF used the same nodes for the second and last passes, and the training data is copied to each node's local disk, it does not have to be copied again in the second or third passes. I.e. GP is run 201 times but the training data have to be copied only 20 times. Each GP run is allocated its own GPU. So there are three GP runs on the three GPU nodes (total 36) and eight GP runs on the eight GPU nodes (64). NFS is used to transfer the 100 output files back to the disk server.

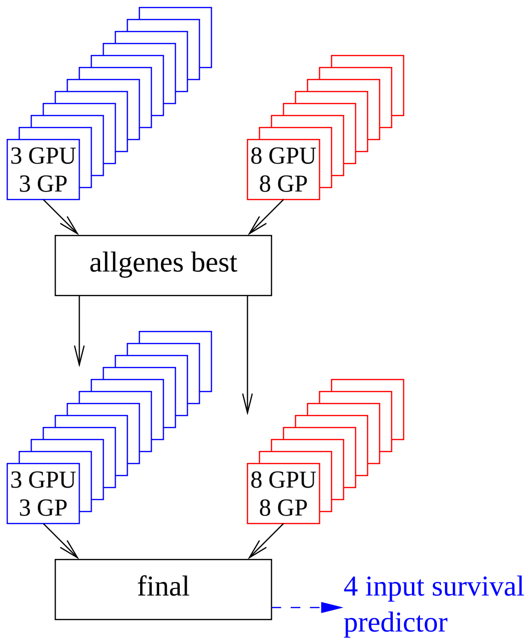


Fig. 3: Schematic of LSF jobs used when extracting predictive genetic factors from the Uppsala breast cancer dataset on the Emerald supercomputer. Each rectangle represents an LSF job containing 0, 3 or 8 GP runs. LSF job array with $12 \times 3$ gpu_gp_cuda runs per node in blue, left. (Total 36 GP runs.) LSF array with $8 \times 8$ gpu_gp_cuda runs per node in red, right. (Total 64 GP runs.) Job `allgenes best` gathers genes from first pass and passes the best onto the second pass. LSF cannot start it until all the first pass jobs are completed (black arrows show control dependencies). When it completes, LSF can start the two second pass job arrays. Similarly LSF cannot start the final job until all the second pass jobs are finished. It gathers genes from the second pass runs before starting the final GP run. Note similarity with data flows shown in Figure 2 and see also the timing diagram in Figure 4.

LSF is used to queue up the small job `allgenes best` which extracts genes used by any of the best models in the final generation of the 100 first pass GP runs. It is simple and does not consume much CPU time (and no GPU time). This list of 28 442 genes is stored on the file server but it is only 222KB and so can be easily read via Emerald's network by the second pass jobs. Effectively `allgenes best` condenses about 1.5MB down to 222 Kbytes. LSF is used to release the job when both first pass job arrays have finished and to start the corresponding two second pass job arrays when it is done. A similar process is used to gather genes selected by the second pass GP runs and start the `final` LSF job containing the final GP run. (For convenience the final gene gathering and the last GP run are combined into a single LSF job.)

The second pass GPU jobs are very similar to first pass and (apart from reading the training data) take about the same time (see Figure 4).

The intention of this approach was to concentrate the task of reading the training data on relatively few nodes and then use all the GPUs on those nodes in parallel. In principle it should have been possible to run all of the 100 GP runs in parallel on 100 GPUs, however since Emerald is a shared with other users this was not possible. As Figure 4 shows although twelve 3 GPU nodes were available only seven 8 GPU nodes were exclusively available. This meant LSF had to reuse one 8 GPU node and the application did not get the full parallelism theoretically available in Emerald. This increased the total duration by about 40% however Emerald still interpreted on average a total of 33.8 giga GPop/S.

## 5 Discussion

There are certainly things that can be done within our application to improve it. At present host operations take a surprisingly large amount of time. For example, selection, crossover and mutation are currently done serially on the host but Pospichal *et al*. [12] have shown, in the case of grammatical evolution, they can be done in parallel on the GPU.

Using 20 nodes is not the only option. LSF is quite flexible. Technical report [7] describes using a single LSF array per pass in which each job contains one GP run. This allows LSF more freedom to schedule jobs and to potentially give the maximum degree of parallelism. LSF can start the next operation when sufficient jobs have successfully completed. If more jobs are started than are needed, this allows some redundancy and error recovery (albeit at the cost of running unneeded jobs).

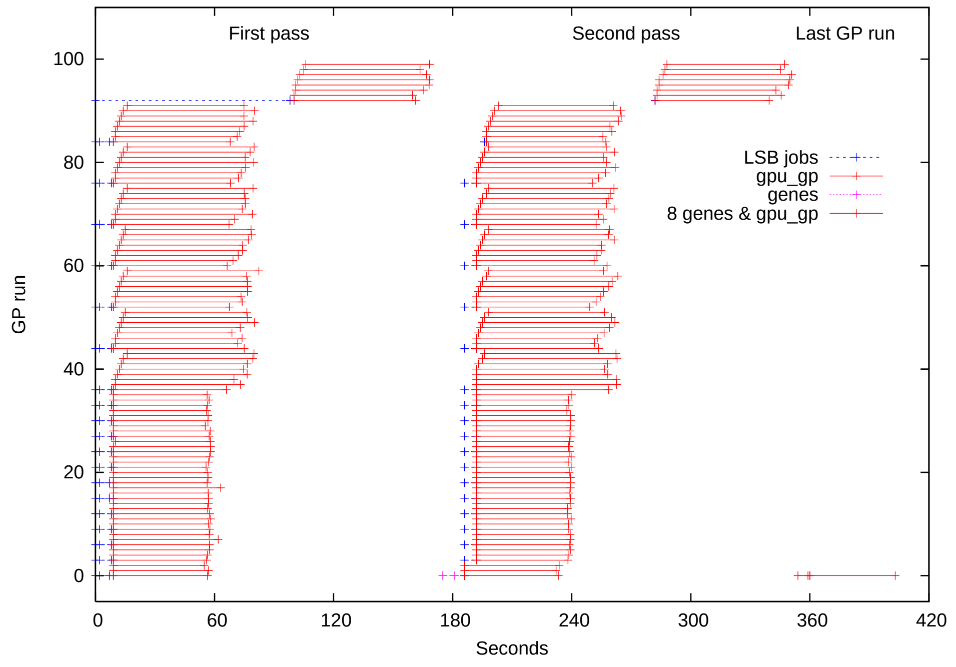


Fig. 4: GP runs to winnow genetic factors for predicting survival after breast surgery from millions of gene expression data. The plot shows parallel running of $2 \times 100 + 1$ GP runs each with a population of five million with all fitness evaluations done on Emerald's M2090 nVidia Tesla cards. The first 1.6 seconds are used copying 352MB of training data to the compute nodes. On the 3 GPU nodes, all three passes take on average 48 seconds. The runs on 8 GPU nodes take 63 seconds on average. (With more GPUs there is more contention for the shared IDE bus.) Only seven 8 GPUs nodes were available, so in both the first and second passes, GP runs 92–99 must wait until others have finished. Gathering and processing all the output generated by each pass and making it ready for the next pass takes about six seconds in both cases. The final GP run finished 6 minutes 43 sec after the whole application was submitted to Emerald during which time 150 billion GP primitives were interpreted on each of 91 training cases, giving a composite rate of 33.8 giga GPop/second.

However, at present, there is no guarantee that once LSF has found a free CPU there will also be a GPU free on the same node. Another approach that I tried was to have a three LSF jobs per pass sent to nodes with eight GPUs. Each job runs GP as many times as needed in parallel on as many GPUs as are free. This readily allows moving GP runs between GPUs if one or more are not available and reduces network traffic between the nodes and Emerald's disk server. Again this is described in [7] but obviously it allows at best only 24 GP runs in parallel rather than potentially all those in the current pass.

Given a task large enough to warrant using all of Emerald, avoiding scheduling difficulties, resolving the CUDA `error 46` issue (Section 3) and doing more of the genetic operations on the GPU, Emerald should also be able to interpret at least half a tera GP operations per second (0.5T GPop/S). However with more than a thousand CPU cores available, it should also be possible to compile GP populations in parallel [1] thus avoiding the interpreter overhead. In principle it also should be possible to evolve binary machine code [10] for graphics cards thus avoiding both the compilation and interpreter overheads. However, so far there has been little progress on this. Nonetheless, Lewis [8] has started in this direction by directly evolving intermediate (assembler like) code.

Datamining the GSE3494 breast cancer dataset requires only about 30 minutes GPU computation on an single M2090. I was worried this would be too small for Emerald. However I have been impressed by the responsiveness of Emerald and the LSF batch system in particular. It appears to react quickly to jobs finishing, is able to quickly start new ones in response and can cope with hundreds of simultaneous jobs.

## 6 Conclusions

An existing CUDA GPGPU genetic programming application which evolved a predictor of long term breast cancer outcomes using Affymetrix gene expression data [3] has been ported to Emerald without code change. Despite Emerald being a shared resource (and therefore the application not being able to access Emerald's full parallelism) the GP interpreter averaged more than 33 billion genetic programming operations per second. This is the fastest floating point genetic programming datamining application so far.

The initial disk server bandwidth problems (Section 3) have essentially been solved. However the problem in which attempts to use a Tesla GPU hardware board are sometimes rejected saying it is in use when it is not (Section 3, `error 46`) and difficulties of reserving parts of Emerald for exclusive use (with `brsvs`) have yet to be resolved. With any new system there are pitfalls for the novice. Some of these are described in technical report [7].

## Acknowledgements

I am grateful for the assistance of Derek Ross, Mike Giles, Andrew Richards, njuffa (nVidia), Gihan R. Mudalige, Juan Bicarregui and Simon McIntosh-Smith.

Funded by EPSRC grant EP/I033688/1. Code developed on GPUs donated by nVidia as part of the gismo project.

## References

[1] Simon L. Harding and Wolfgang Banzhaf. Distributed genetic programming on GPUs using CUDA. In Ignacio Hidalgo, Francisco Fernandez, and Juan Lanchares, editors, *Workshop on Parallel Architectures and Bioinspired Algorithms*, pages 1–10, Raleigh, NC, USA, 13 September 2009. Universidad Complutense de Madrid.

[2] Andrew P. Harrison, Joanna Rowsell, Renata da Silva Camargo, William B. Langdon, Maria Stalteri, Graham J.G. Upton, and Jose M. Arteaga-Salas. The use of Affymetrix GeneChips as a tool for studying alternative forms of RNA. *Biochemical Society Transactions*, 36:511–513, 2008.

[3] W. B. Langdon. Large scale bioinformatics data mining with parallel genetic programming on graphics processing units. In S. Tsutsui and P. Collet, editors, *Evolutionary Computation on Graphics Processing Units*, chapter 20. Springer.

[4] W. B. Langdon and B. F. Buxton. Genetic programming for mining DNA chip data from cancer patients. *Genetic Programming and Evolvable Machines*, 5(3):251–257.

[5] W. B. Langdon and A. P. Harrison. GP on SPMD parallel graphics hardware for mega bioinformatics data mining. *Soft Computing*, 12(12):1169–1183, October 2008.

[6] W. B. Langdon, G. J. G. Upton, R. da Silva Camargo, and A. P. Harrison. A survey of spatial defects in Homo Sapiens Affymetrix GeneChips. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 7(4):647–653, oct.-dec 2009.

[7] W. B. Langdon. Initial experiences of the emerald: e-infrastructure south GPU supercomputer. Research Note RN/12/08, Department of Computer Science, University College London, Gower Street, London WC1E 6BT, UK, 17 June 2012.

[8] Tony E. Lewis and George D. Magoulas. TMBL kernels for CUDA GPUs compile faster using PTX. In Simon Harding, W. B. Langdon, Man Leung Wong, Garnett Wilson, and Tony Lewis, editors, *GECCO 2011 Computational intelligence on consumer games and graphics hardware (CIGPU)*, pages 455–462, Dublin, Ireland, 12-16 July 2011. ACM.

[9] Lance D. Miller, *et al.* An expression signature for p53 status in human breast cancer predicts mutation status, transcriptional effects, and patient survival. *Proceedings of the National Academy of Sciences*, 102(38):13550–5, Sep 20 2005.

[10] Peter Nordin. A compiling genetic programming system that directly manipulates the machine code. In Kenneth E. Kinnear, Jr., editor, *Advances in Genetic Programming*, chapter 14, pages 311–331. MIT Press, 1994.

[11] Riccardo Poli, William B. Langdon, and Nicholas Freitag McPhee. *A field guide to genetic programming*. Published via `http://lulu.com` and freely available at `http://www.gp-field-guide.org.uk`, 2008. (With contributions by J. R. Koza).

[12] Petr Pospichal, Eoin Murphy, Michael O'Neill, Josef Schwarz, and Jiri Jaros. Acceleration of grammatical evolution using graphics processing units: computational intelligence on consumer games and graphics hardware. In Simon Harding, W. B. Langdon, Man Leung Wong, Garnett Wilson, and Tony Lewis, editors, *GECCO 2011 Computational intelligence on consumer games and graphics hardware (CIGPU)*, pages 431–438, Dublin, Ireland, 12-16 July 2011. ACM.

## About the author

Dr. Langdon was a research officer at the Central Electricity Generating Board's research laboratories and then a software consultant with Logica before returning to academic life and gaining a PhD at University College, London in genetic programming (GP). He has recently joined the CREST software engineering center which applies artificial intelligence techniques to program production, testing, debugging and maintenance. Recently Bill has been applying genetic programming to automatically generate interfaces (GIPs) and other small but critical parts of larger software systems.

Homepage: http://www.cs.ucl.ac.uk/staff/W.Langdon/
Email: W.Langdon@cs.ucl.ac.uk

# Calls and Calendar

## January 2013

**Learning and Intelligent OptimizationN Conference - LION 6**
January 7-11, 2013, Catania, Italy
Homepage: http://www.intelligent-optimization.org/LION7/
Call for Papers: www
Deadline October 14, 2012
Conference dates: January 7-11, 2013

The large variety of heuristic algorithms for hard optimization problems raises numerous interesting and challenging issues. Practitioners are confronted with the burden of selecting the most appropriate method, in many cases through an expensive algorithm configuration and parameter tuning process, and subject to a steep learning curve. Scientists seek theoretical insights and demand a sound experimental methodology for evaluating algorithms and assessing strengths and weaknesses. A necessary prerequisite for this effort is a clear separation between the algorithm and the experimenter, who, in too many cases, is "in the loop" as a crucial intelligent learning component. Both issues are related to designing and engineering ways of "learning" about the performance of different techniques, and ways of using past experience about the algorithm behavior to improve performance in the future. Intelligent learning schemes for mining the knowledge obtained from different runs or during a single run can improve the algorithm development and design process and simplify the applications of high-performance optimization methods. Combinations of algorithms can further improve the robustness and performance of the individual components provided that sufficient knowledge of the relationship between problem instance characteristics and algorithm performance is obtained.

This meeting, which continues the successful series of LION events (see LION 4 at Venice, and LION 5 at Rome, and LION 6 at Paris), is aimed at exploring the intersections and uncharted territories between machine learning, artificial intelligence, mathematical programming and algorithms for hard optimization problems. The main purpose of the event is to bring together experts from these areas to discuss new ideas and methods, challenges and opportunities in various application areas, general trends and specific developments.

LION 7 Conference and Technical co-chairs

- Panos Pardalos, University of Florida (USA)
- Giuseppe Nicosia, University of Catania (Italy)

## April 2013

**Evostar 2012 - EuroGP, EvoCOP, EvoBIO, EvoMusart and EvoApplications**
April 3-5, 2013, Vienna, Austria
Homepage: http://www.evostar.org
Deadline November 1, 2012
Camera-ready deadline: January 15, 2013

**EvoStar** comprises of five co-located conferences run each spring at different locations throughout Europe. These events arose out of workshops originally developed by EvoNet, the Network of Excellence in Evolutionary Computing, established by the Information Societies Technology Programme of the European Commission, and they represent a continuity of research collaboration stretching back nearly 20 years.

**EuroGP (www)**

16th European Conference on Genetic Programming Papers are sought on topics strongly related to the evolution of computer programs, ranging from theoretical work to innovative applications.

**EvoBIO (www)**

11th European Conference on Evolutionary Computation, Machine Learning and Data Mining in Computational Biology Emphasis is on evolutionary computation and other advanced techniques addressing important problems in molecular biology, proteomics, genomics and genetics, that have been implemented and tested in simulations and on real-life datasets.

**EvoCOP (www)**

13th European Conference on Evolutionary Computation in Combinatorial Optimization Practical and theoretical contributions are invited, related to evolutionary computation techniques and other meta-heuristics for solving combinatorial optimization problems.

**EvoMUSART (www)**

2nd International Conference (and 11th European Event) on Evolutionary and Biologically Inspired Music, Sound, Art and Design.

**EvoApplications (www)**

15th European Conference on the Applications of Evolutionary Computation

- **EvoCOMNET (www)**: Application of Nature-inspired Techniques for Communication Networks and other Parallel and Distributed Systems
- **EvoCOMPLEX (www)**: Applications of algorithms and complex systems
- **EvoENERGY (www)**: Evolutionary Algorithms in Energy Applications
- **EvoFIN (www)**: Track on Evolutionary Computation in Finance and Economics
- **EvoGAMES (www)**: Bio-inspired Algorithms in Games
- **EvoIASP (www)**: Evolutionary computation in image analysis, signal processing and pattern recognition
- **EvoINDUSTRY (www)**: The application of Nature-Inspired Techniques in industrial settings
- **EvoNUM (www)**: Bio-inspired algorithms for continuous parameter optimisation
- **EvoPAR (www)**: Parallel and distributed Infrastructures
- **EvoRISK (www)**: Computational Intelligence for Risk Management, Security and Defense Applications
- **EvoROBOT (www)**: Evolutionary Computation in Robotics
- **EvoSTOC (www)**: Evolutionary Algorithms in Stochastic and Dynamic Environments

## July 2013

**GECCO 2013 - Genetic and Evolutionary Computation Conference**

July 6-10, 2013, Amsterdam, The Netherlands

Homepage: http://www.sigevo.org/gecco-2013

Deadline January 23, 2013

Workshop and tutorial proposals submission: November 07, 2012

Author notification: March 14, 2013

The Genetic and Evolutionary Computation Conference (GECCO-2013) will present the latest high-quality results in the growing field of genetic and evolutionary computation.

Topics include: genetic algorithms, genetic programming, evolution strategies, evolutionary programming, real-world applications, learning classifier systems and other genetics-based machine learning, evolvable hardware, artificial life, adaptive behavior, ant colony optimization, swarm intelligence, biological applications, evolutionary robotics, coevolution, artificial immune systems, and more.

### Important Dates

| | |
|---|---|
| Paper Submission Deadline | January 23, 2013 |
| Decision Notification | March 14, 2013 |
| Camera-ready Submission | April 17, 2013 |

### Organizers

| | |
|---|---|
| General Chair: | Enrique Alba |
| Editor-in-Chief: | Christian Blum |
| Proceeding Chair: | Leonardo Vanneschi |
| Local Chairs: | Peter Bosman |
| | Evert Haasdijk |
| Publicity Chair: | Xavier Llorá |
| Tutorials Chair: | Gabriela Ochoa |
| Students Chair: | Emilia Tantar |
| Workshops Chair: | Mike Preuss |
| Competitions Chairs: | Daniele Loiacono |
| Business Committee: | Darrell Whitley |
| | Marc Schoenauer |
| EC in Practice Chairs: | Jörn Mehnen |
| | Thomas Bartz-Beielstein, |

### How to Submit a Paper

Meet the submission deadline - January 23, 2013 - and submit substantially new work. GECCO allows submissions of material that is substantially similar to a paper being submitted contemporaneously for review in another conference. However, if the submitted paper is accepted by GECCO, the authors agree that substantially the same material will not be published by another conference in the evolutionary computation field. Material may be later revised and submitted to a journal, if permitted by the journal.

### More Information

Visit www.sigevo.org/gecco-2013 for information about deadlines, student travel grants, hotel reservations, student housing, the graduate student workshop, the latest list of topics, late-breaking papers, and more. For matters of science and program content, contact Conference Chair Enrique Alba at gecco2013chair@sigevolution.org while for general help and administrative matters please contact GECCO support at gecco2013@sigevolution.org

GECCO is sponsored by the Association for Computing Machinery Special Interest Group for Genetic and Evolutionary Computation.

# About the Newsletter

SIGEVOlution is the newsletter of SIGEVO, the ACM Special Interest Group on Genetic and Evolutionary Computation.

To join SIGEVO, please follow this link [WWW]

## Contributing to SIGEVOlution

We solicit contributions in the following categories:

**Art**: Are you working with Evolutionary Art? We are always looking for nice evolutionary art for the cover page of the newsletter.

**Short surveys and position papers**: We invite short surveys and position papers in EC and EC related areas. We are also interested in applications of EC technologies that have solved interesting and important problems.

**Software**: Are you are a developer of an EC software and you wish to tell us about it? Then, send us a short summary or a short tutorial of your software.

**Lost Gems**: Did you read an interesting EC paper that, in your opinion, did not receive enough attention or should be rediscovered? Then send us a page about it.

**Dissertations**: We invite short summaries, around a page, of theses in EC-related areas that have been recently discussed and are available online.

**Meetings Reports**: Did you participate in an interesting EC-related event? Would you be willing to tell us about it? Then, send us a short summary, around half a page, about the event.

**Forthcoming Events**: If you have an EC event you wish to announce, this is the place.

**News and Announcements**: Is there anything you wish to announce? This is the place.

**Letters**: If you want to ask or to say something to SIGEVO members, please write us a letter!

**Suggestions**: If you have a suggestion about how to improve the newsletter, please send us an email.

Contributions will be reviewed by members of the newsletter board.

We accept contributions in LaTeX, MS Word, and plain text.

Enquiries about submissions and contributions can be emailed to editor@sigevolution.org.

All the issues of SIGEVOlution are also available online at www.sigevolution.org.

## Notice to Contributing Authors to SIG Newsletters

By submitting your article for distribution in the Special Interest Group publication, you hereby grant to ACM the following non-exclusive, perpetual, worldwide rights:

- to publish in print on condition of acceptance by the editor
- to digitize and post your article in the electronic version of this publication
- to include the article in the ACM Digital Library
- to allow users to copy and distribute the article for noncommercial, educational or research purposes

However, as a contributing author, you retain copyright to your article and ACM will make every effort to refer requests for commercial use directly to you.